Universitat de Lleida
Escola Politècnica Superior
Enginyeria Tècnica en Informàtica de Sistemes

Treball de final de carrera

CompP2P-GUI: Interfície gràfica del Sistema de Computació Distribuïda Peer to Peer (CompP2P)

Autor: Ignasi Barri Vilardell

Directors: Francesc Solsona Tehàs
Fernando Cores Prado

Febrer de 2007

Attribution-ShareAlike 2.5

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

   (a) "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

   (b) "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.

   (c) "Licensor" means the individual or entity that offers the Work under the terms of this License.

   (d) "Original Author" means the individual or entity who created the Work.

(e) "Work" means the copyrightable work of authorship offered under the terms of this License.

(f) "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

(g) "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

(a) to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

(b) to create and reproduce Derivative Works;

(c) to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

(d) to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works.

(e) For the avoidance of doubt, where the work is a musical composition:

i. Performance Royalties Under Blanket Licenses. Licensor waives the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work.

ii. Mechanical Rights and Statutory Royalties. Licensor waives the exclusive right to collect, whether individually or via a music rights society or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions).

(f) Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor waives the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions).

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions.The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

(a) You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by clause 4(c), as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any credit as required by clause 4(c), as requested.

(b) You may distribute, publicly display, publicly perform, or publicly digitally perform a Derivative Work only under the terms of this License, a later version of this License with the same License Elements as this License, or a Creative Commons iCommons license that contains the same License Elements as this License (e.g. Attribution-ShareAlike 2.5 Japan). You must include a copy of, or the Uniform Resource Identifier for, this License or other license specified in the previous sentence with every copy or phonorecord of each Derivative Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Derivative Works that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder, and You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Derivative Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Derivative Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Derivative Work itself to be made subject to the terms of this License.

(c) If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit

appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer

   UNLESS OTHERWISE AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE MATERIALS, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   (a) This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   (b) Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   (a) Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   (b) Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

   (c) If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

   (d) No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

(e) This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

Pròleg

CompP2P és una plataforma la potència de la qual rau en la quantitat d'equips que en fas ús, quant més usuaris cedeixin la seva potència de càlcul inutilitzada, més potent serà el sistema. Per tant, cal fer arribar al màxim nombre de persones possible el programa, així com facilitar el seu ús. És en aquest moment quan pren vital importància la creació d'un entorn d'usuari amigable.

Com s'ha vist en experiments sobre *CompP2P*, la potencia d'aquesta plataforma rau en la capacitat de fragmentació de la feina i no tant en la potencia de càlcul per cadascuna de les tasques. Amb una política de repartició de feina que tingui en compte la productivitat dels nodes per fer una assignació intel·ligent, tindrà més possibilitats de fer un balanceig efectiu amb un numero elevat de subtasques a realitzar.

En la situació actual, els usuaris potencials d'una plataforma per consola amb una execució de treballs per text és escassa, una plataforma d'aquest tipus no arribaria a un sector suficientment ampli com per dotar el programa d'una distribució de les tasques suficients, tornant així al paradigma dels supercomputadors. Fent arribar l'eina als usuaris d'una forma més visual, permetria canviar les tornes i provocaria que l'usuari mig pogués executar el programa en el seu ordinador, ampliant així, el ventall d'usuaris finals de l'aplicatiu.

Vista la problemàtica i la seva solució, en aquest treball final de carrera s'ha desenvolupat un entorn d'usuari que gestioni totes les funcionalitats de *CompP2P*, a més de la introducció de millores en la plataforma de comunicació per permetre una millor interoperabilitat amb el sistema.

Anteriorment, la gestió de la plataforma de còmput distribuït podia resultar farragosa per un usuari final. Amb la introducció d'aquesta interfície gràfica un usuari qualsevol pot posar en marxar un node o treure'l en un parell de clicks. També facilita el llançament de tasques, així com la obtenció d'informació de forma ràpida i intuïtiva.

Al igual que el sistema base, l'aplicació es independent del sistema operatiu, el que com és l'objectiu de la interficie permet arribar a més usuaris. Això s'aconsegueix mitjançant la comunicació per xarxa de CompP2P. Derivada d'aquesta darrera característica, s'ha dotat el sistema de gestió remota, permetent així un control de diversos equips, agilitzant així la tasca de control dels nodes de la plataforma.

Íñigo Goiri Presa

Índex

Índex de figures

Capítol 1

Introducció

La quantitat de càlculs que es fan a diari són enormes. Les ciències són les que precisen un rendiment computacional més elevat, per tal de solucionar problemes que són computacionalment molt complexes i amb requeriments de dades molt elevats. Aquests càlculs es realitzen en un ventall molt ampli de ciències (biotecnologia, matemàtica, climatologia, física...), són càlculs que no es poden resoldre si es fa ús d'un únic computador.

Usar supercomputadors per aquestes tasques és ineficient, ja que són poc escalables i tenen costos econòmics molt elevats. Per posar remei a aquest fet, han sorgit dues branques en computació distribuïda; aquestes són:

1. Emular un supercomputador a partir d'un conjunt heterogeni d'ordinadors connectats entre si mitjançant una *LAN* (Local Area Network).

2. Utilitzar molts computadors domèstics connectats a través d'Internet simulant així un supercomputador.

La interfície que es desenvoluparà en aquest treball és la d'una aplicació que es troba dins la segona opció de les dos que acabem de comentar. Aquests sistemes ens permeten assolir objectius més ambiciosos, doncs la potència de càlcul és elevada i la quantitat de recursos dels que es disposa és molt ampli.

Un sistema de càlcul que enllaci molts computadors connectats entre sí és molt potent, per què? Per exemple, avui en dia els ordinadors personals que estan en el mercat diposen d'unes característiques tècniques molt superiors a les que els usuaris veritablement necessiten. Ordenadors amb 3GHz de microprocessador i 1GB de memòria *RAM*, són malaprofitats per gran part dels seus usuaris, ja que utilitzen mínimament aquests potencial, desaprofitant, doncs, tota la capacitat de còmput o recursos totals del sistema. Fent un càlcul ràpid podem veure que si posem en comú els recursos de 100 màquines amb les característiques que hem comentat, obtindríem un macroprocessador de 225GHz i una memòria *RAM* de 75GB. Es pot apreciar, que si el nombre d'ordinadors creix, es pot arribar a uns recursos computacionals exorbitants. La interfície gràfica que recobreix el projecte *CompP2P* és una intefície d'una aplicació que intenta aprofitar tots aquests recursos de còmput disponibles.

Aquesta potència de càlcul però, no és del tot real, doncs si els equips estan connectats entre ells, hi haurà una pèrdua d'efectivitat degut a Internet ja que és el sistema de comunicació entre les màquines, doncs l'aplicació estarà molt més temps esperant rebre les dades necessàries i enviant el resultats dels còmputs realitzats.

El sistema *CompP2P* aprofita la capacitat de còmput dels ordinadors que es troben connectats en una xarxa *LAN*.

En el projecte desenvolupat per **Josep Rius Torrentó** i **Íñigo Goiri Presa** en el seu *Treball Final de Carrera*[10] s'expliquen tots els conceptes necessaris per poder entendre sense problemes l'aplicatiu *CompP2P*.

El treball que he realitzat es l'interfície gràfica d'usuari (*GUI*[1]) de l'aplicació *CompP2P*, és a dir, consisteix en un aplicatiu que proporciona un entorn fàcil i amigable que serveix d'interfície amb el sistema complet.

CompP2P és una aplicació de computació distribuida p2p que té moltes possibilitats en el camp de la recerca, ja que permet muntar fàcilment una xarxa de còmput molt potent a més a més sense cap cost econòmic addicional. L'inconvenient que té aquesta aplicació, és la dificultat d'ús o de gestió de la mateixa, ja que conéixer el seu funcionament requereix tenir uns coneixements de l'aplicació massa propers a com aquesta està estructurada o com funciona internament.

Aquest punt, comdemna a *CompP2P* a un camí sense sortida, on la manca de simplicitat alhora d'usar l'aplicació, provocarà que els usuaris no usin el sistema degut a que desconeixen com usar-lo, fet que els portarà a escollir altres aplicacions més usables. D'aquesta necessitat d'oferir quelcom més que una aplicació de còmput distribuït, neix l'aplicació *CompP2P-GUI*, una interfície gràfica que permet la gestió i el maneig de *CompP2P* de forma intuitiva i senzilla.

La interfície gràfica és un tipus d'interfície d'usuari que sorgeix de l'evolució de la línia de comandes dels primers sistemes operatius i és una peça fonamental d'un entorn gràfic. Aquests tipus d'interfícies utilitzen un conjunt d'imatges i objectes gràfics (icones, finestres, tipografia) per representar la informació i accions disponibles en l'aplicació. Habitualment les accions es realitzen mitjançant la manipulació directa per facilitar la interacció de l'usuari amb la seva màquina.

En els primer capítol d'aquest treball s'efectua una introducció a la filosofia *Peer-to-Peer*. Primerament, es farà una petit estudi de les plataformes *P2P* de còmput distribuït. A continuació s'analitzarà la gestió i la monitorització d'altres sistemes *P2P* com són: *gnutella*, *amule*, *azureus*... etc.

1.1 Computació distribuïda P2P

En aquesta secció, és farà una breu repassada als sistemes de còmput distribuït *peer-to-peer*.

1.1.1 Paradigma P2P

Un paradigma de còmput que ha sorgit recentment és la computació distribuïda peer-to-peer[13, 14, 15, 16, 17] que estan basats en xarxes d'ordenadors entre iguals. Una xarxa entre iguals (p2p) fa referència a una xarxa que no té ni servidors ni clients fixes, sinó una sèrie de nodes que es comporten simultàniament com clients i com servidors de la resta de nodes de la xarxa. Aquest model de xarxa contrasta amb el model client-servidor el qual va regit d'una arquitectura monolítica on no hi ha distribució de tasques entre sí, només una simple comunicació entre un usuari i una terminal on el client i el servidor no poden canviar els rols.

En el model P2P cada node connectat a la xarxa (peer) té capacitats i responsabilitats equivalents alhora d'utilitzar i compartir recursos distribuits per la realització d'una funció de forma descentralitzada. Degut a que els sistemes P2P involucren la participació dels nodes d'una xarxa sense administració central, s'han de considerar les següents característiques en aquests entorns:

1. Connectivitat de xarxa variable seguint qualsevol tipologia, ja que els nodes poden connectar-se i desconnectar-se d'aquesta de forma arbitrària.

[1] Graphic User Interface.

2. Ubicació dinàmica dels serveis i recursos.
3. Auto-organització, cada node regula el seu grau de participació en la xarxa.
4. Un entorn altament dinàmic i heterogeni.

La selecció del paradigma de còmput P2P es fonamenta en els avantatges que aporta:

- **Escalabilitat il·limitada**, degut a la no existència de components centralitzats que actuen com un coll de botella.
- **Tolerància a errades**. L'alt dinamisme del sistema obliga a que estigui contínuament funcionant encara que hi hagin moltes errades i desconnexions de nodes del sistema.
- **Altes prestacions potencials** a nivell de recursos ociosos susceptibles d'ésser gestionats i compartits pel sistema.
- **baix cost**, ja que es basa en la compartició de recursos ja existents i no dedicats.

En contrast a les avantatges mencionats, les arquitectures P2P tenen una sèrie de característiques intrínseques que les diferencïen d'altres alternatives de còmput ditribuït i que han de tenir-se en compte en la consecució dels objectius plantejats pel projecte[28]:

- **Heterogeneïtat del sistema**. El sistema P2P està composat per un conjunt de nodes de diferents característiques (processador, sistema operatiu, memòria ...) connectats entre sí per diferents tipologies de xarxa (fixa, dinàmica, inalàmbrica) i de diferents àmbits (local,global), latència i ample de banda variables.
- **Recursos no dedicats**. Els recursos de còmput estan compartits per usuaris globals i usuaris locals. La utilització dels recursos per part dels usuaris globals no ha d'afectar a les prestacions que obtenen els usuaris locals de cada màquina connectada al sistema sobre la qual tenen prioritat.
- **Dinamisme**. El sistema està format per ordenadors no dedicats i que no estan permanentment connectats. Això provoca una entrada i sortida dels nodes del sistema de forma arbitrària que introdueix una alta variabilitat en la configuració del mateix i volatilitat en els recursos.

1.2 Context actual

A continuació, es farà un petit sumari de l'estat actual dels dos elements fonamentals del projete: els sistemes *P2P* així com de les interfícies gràfiques *GUI*.

1.2.1 Punt de partida dels sistemes *P2P*

Actualment, hi ha molts sistemes *P2P*. La gran majoria són d'intercanvi d'arxius entre usuari en el que regna la filosofia *P2P*, aquesta filosofia estableix un sistema meritocràtic[2] on "el que més comparteix, més privilegis té i disposa de més accés a més contingut de manera més ràpida". Amb aquests sistema s'intenta assegurar la disponibilitat del contingut compartit, ja que en cas contrari no seria possible la subsistència de la xarxa.

[2]Meritocràcia (del llatí mereo, meréixer, obtenir) és una forma de govern basada en el mèrit. Les posicions jeràrquiques són consquistades en base al mèrit obtingut, hi ha un predomini de valors associats a l'educació i a la competència.

Aquests sistemes, però, no són massa freqüents en el camp del còmput, per tant és necessari i convenient que s'efectuïn estudis per tal d'ampliar el camp de les aplicacions *P2P* dirigides al desenvolupament de projectes que requereixen capacitat de còmput.

CompP2P pretén donar una alternativa diferent a la computació distribuïda que s'utilitza actualment, i a les aplicacions típiques de *P2P*, creant un sistema de fàcil ús per qualsevol equip del món, sense necessitat de tenir gaires coneixements en el camp per posar-lo en marxa. Aconseguint, d'aquesta manera, un sistema totalment entre iguals, i usable per tothom.

1.2.2 Punt de partida de les interfícies gràfiques *GUI*

La història de la informàtica es troba indissolublement unida a les interfícies gràfiques, ja que els sistemes operatius gràfics han tingut grans ampliacions en la indústria del *software* i del *hardware*.

Les interfícies gràfiques sorgeixen de la necessitat de fer els ordinadors més accessibles per l'ús dels usuaris comuns. La gran limitació que tenien les computadores fins el moment pel seu ús massiu i extès, era que només funcionaven per línia de comanda, el que requeria que la persona que vulgués utilitzar la màquina havia de tenir un mínim coneixement sobre el seu funcionament.

Aquesta limitació es va superar gràcies al desenvolupament dels entorns gràfics, que van permetre que les persones puguessin accedir a un ordinador sense tenir que passar per processos d'aprenentatge d'un entorn de línia de comandes que resultaven complicats d'assimilar

CompP2P-GUI pretén facilitar l'ús de *CompP2P* per tal d'ampliar el nombre d'usuaris potencials de l'aplicació primitiva, ja que permetrà posar en funcionament un sistema de còmput *P2P* a persones que no tinguin coneixements avançats d'arquitectures distribuïdes així com de coneixements extensos en el propi sistema.

1.3 Aplicacions *P2P*: estudi i anàlisi

1.3.1 Inicis de les aplicacions *P2P*

La primera aplicació *P2P* (Peer-to-Peer, o entre parells), fou *Napster* l'any 1999. La transferència d'arxius es porten a terme entre dos equips. *Napster* utilitzava servidors centrals per emmagatzemar la llista d'equips i els arxius que proporcionava cada un d'ells, per tant no era una aplicació purament *P2P*. Per aquells temps, però, ja existien aplicacions que permitien l'intercanvi entre usuaris, com el *IRC* i *Usenet*.

Degut a que la majoria dels ordinadors domèstics no tenen una adreça *IP* fixa, sinó que aquesta és assignada per un proveïdor (*ISP*[3]) en el moment de connectar-se a internet, no es possible la connexió entre usuaris de forma directa, doncs els equips no coneixen les adreces dels peers remots que han d'usar amb previsió. Per aquesta raó *Napster* usava uns servidors centrals (amb direcció *IP* fixa), als quals els usuaris es connectaven per tal que els subministrés la relació de direccions *IP* dels clients de la xarxa, de la resta de servidors de la xarxa i d'altra informació referent a la quantitat d'arxius que disposen els clients. Després d'aquesta connexió als servidors centrals, els clients ja disposen d'informació suficient de la xarxa i per tant ja es troben en condició d'intercanviar arxius entre sí, ara ja sense necessitat de cap intervenció per part dels servidors.

Posar fi a les xarxes centralitzades per part d'organismes privats o públics (discogràfiques... etc) era relativament senzill, només era necessari tancar els servidors que emmagatzemava les llistes d'usuaris i arxius compartits. Després de la clausura de les aplicacions d'aquest estil, van sorgir-ne de noves i més

[3]Internet Service Provider o proveïdor de serveis d'internet, és una empresa que es dedica a a connectar a internet els usuaris de diferents xarxes que tinguin i de donar també un servei tècnic als mateixos.

modernes, que van concebre xarxes descentralitzades que no disposaven de servidors centrals i per tant sense que hagués constància dels arxius intercanviats. Aquestes noves tipologies van provocar un avanç tecnològic molt important i significatiu.

1.3.2 Classificació de les xarxes *P2P*

Les xarxes *P2P* es poden classificar segons el seu grau de centralització; així doncs tindrem:

- Xarxes *P2P* centralitzades: Aquest tipus de xarxa *P2P* es basa en una arquitectura monolítica on totes les transaccions es fan a través d'un únic servidor que serveix de punt d'enllaç entre dos nodes. Aquest servidor a la vegada emmagatzema i distribueix els nodes on s'emmagatzemen els continguts. Posseeixen una administració molt dinàmica i una disposició més permanent del contingut encara que està molt limitada pel que fa a la privacitat dels usuaris, l'escalabilitat d'un únic servidor, a més a més d'oferir problemes en punts únics d'errada, situacions legals desfavorables i enormes costos en el manteniment així com del consum de l'ample de banda.

  Les xarxes d'aquest tipus es caracteritzen per:

  – Es controla des d'un únic servidor com punt d'enllaç entre nodes i com un servidor d'accés al contingut, el qual es distribueix a petició dels nodes.
  – Totes les comunicacions (com les peticions i encaminaments entre nodes) depenent exclusivament de l'existència del servidor; si aquest cau, el sistema falla.

  Alguns exemples de xarxes *P2P* centralitzades són: *Napster* i *Audiogalaxy*.

  En la figura 1.1 es pot veure un esquema d'una xarxa *P2P* de tipus centralitzat.

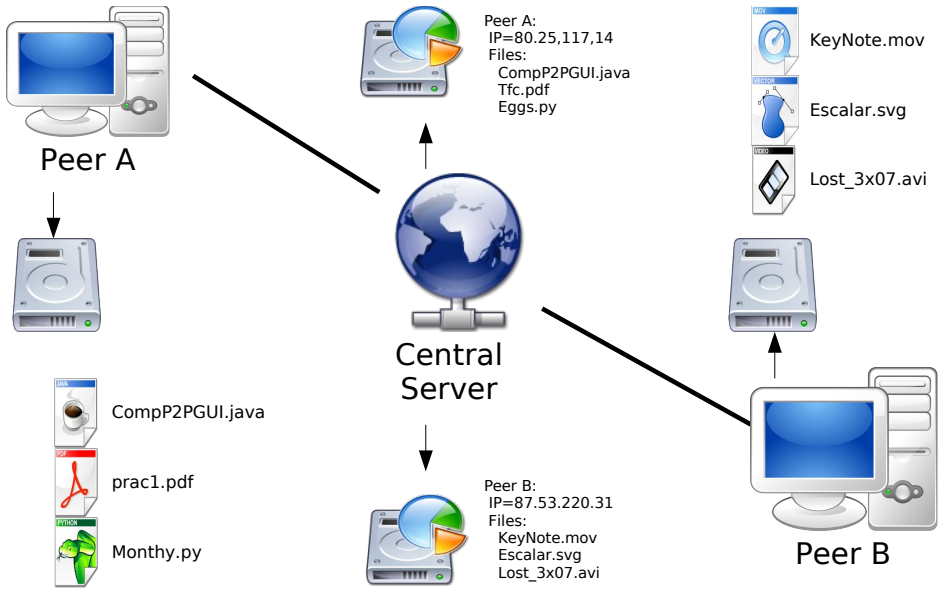

Figura 1.1: Esquema d'una xarxa *P2P* de tipus centralitzada.

- Xarxes *P2P* "pures" o totalment descentralitzades: Aquest tipus de xarxa *P2P* són les més comuns, essent les més versàtils al no requerir d'un gestionament central de cap tipus. Això permet una

reducció de la necessitat d'usar un servidor central per substituïr-lo pels mateixos usuaris com nodes d'aquestes connexions i també com magatzems d'aquesta informació. Amb altres paraules, totes les comunicacions es realitzen directament d'usuari a usuari amb l'ajut d'un node (que es tracta d'un altre usuari) que permet enllaçar aquestes comunicacions.

Les xarxes d'aquest tipus es caracteritzen per:

- Els nodes actuen com client i servidor.
- No existeix un servidor central que gestioni les connexions de la xarxa.
- No hi ha un enrutador central que servieixi com un node i administri les adreces.

Alguns exemples de xarxes *P2P* descentralitzades "pures" són: *Ares Galaxy*, *Gnutella*, *Freenet* i *Gnutella2*.

En la figura 1.3 es pot veure un esquema d'una xarxa *P2P* de tipus centralitzat.

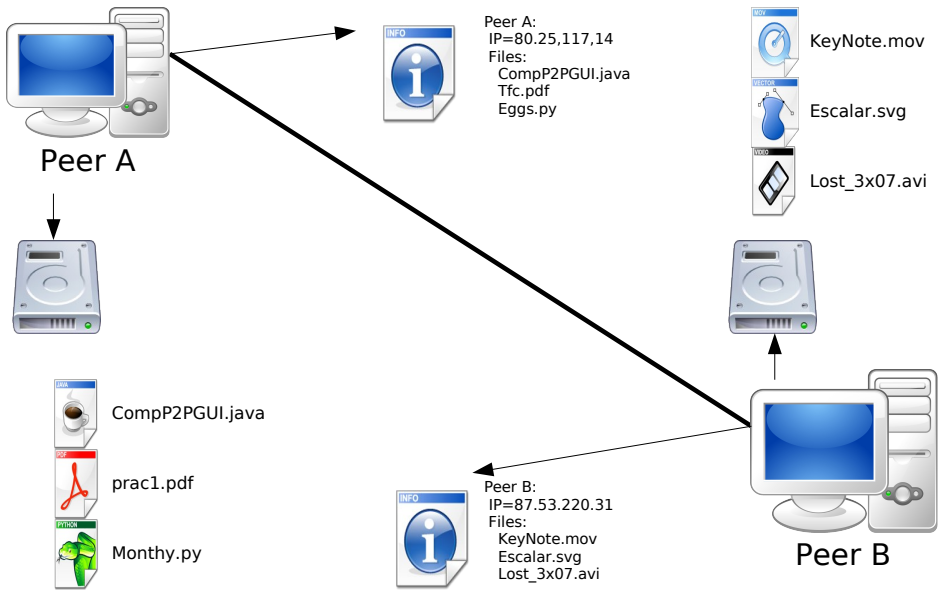

Figura 1.2: Esquema d'una xarxa *P2P* de tipus descentralitzada.

- Xarxes *P2P* híbrides, semi-centralitzades o mixtes: En aquest cas el servidor no coneix la identitat de cada node i tampoc emmagatzema cap tipus d'informació, per tant el servidor no comparteix cap tipus d'arxiu amb cap node. Té la pecurialitat de funcionar en alguns casos (com pot se *Torrent*) d'ambdues maneres, és a dir que pot incorporar més d'un servidor que gestiona els recursos compartits, però també en cas que el o els servidor(s) que gestionen caiguin, el grup de nodes segueix en contacte a través d'una connexió directa entre ells mateixos amb lo qual és possible seguir compartint i descarregant més informació en aubsència dels servidors.

  Les xarxes d'aquests tipus es caracteritzen per:

  - Tenen un servidor central que desa informació i romandrà en espera per respondre les peticions que se li realitzen d'aquesta informació emmagatzemada.

– Els nodes són responsables d'hospedar la informació (ja que el servidor central no ho fa), que permet al servidor central reconéixer els recursos que desitgen compartir, per poder decarregar aquests recursos als peers que ho sol·liciten.
– Les terminals d'enrutament són adreces usades pel servidor, que són administrades per un sistema d'índex per obtenir una adreça absoluta.

Alguns exemples d'una xarxa *P2P* híbrida són: *BitTorrent*, *eDonkey2000* i *Direct Connect*.

En la figura 1.3 es pot veure un esquema d'una xarxa *P2P* de tipus centralitzat.

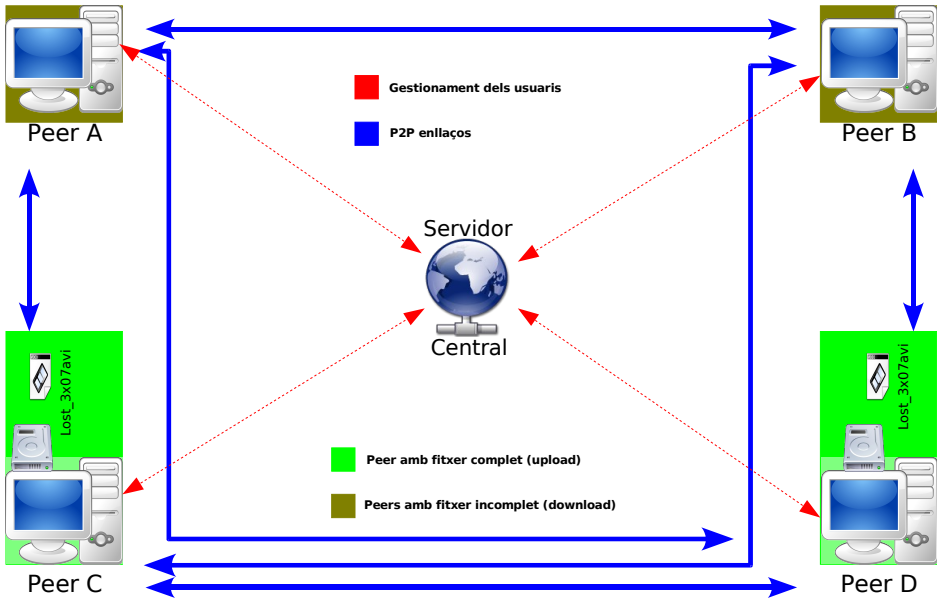

Figura 1.3: Esquema d'una xarxa *P2P* de tipus híbrida o mixta.

1.3.3 Avantatges de les xarxes *P2P*

Un objectiu important en les xarxes *P2P* és que tots els clients que es connecten a ella, són proveïdors de recursos (inclòs banda ampla, espai d'emmagatzemament i processament de còmput). Així doncs, quan els nodes es van connectant, aquestos normalment sol·liciten recursos, aquest augment de recursos provoca que els totals del sistema també augmentin. Aquest fet és diferent en una arquitectura servidor-client com un sistema fixes de servidors, en els quals l'addició de més clients podria significar una transferència de dades més lenta per tots els usuaris.

La naturalesa distribuïda de les xarxes *P2P* també incrementa la robustesa en cas de l'aparició d'errades en la rèplica excessiva de dades cap a múltiples destinacions, i -en sistemes *peer-to-peer* purs- permetint als peers trobar la informació sense necessitat d'efectuar cap petició a cap servidor centralitzat indexat. En aquest darrer cas, no hi ha cap punt singular de fallada en el sistema.

1.3.4 Xarxa de sobrecapa en *P2P*

la xarxa de sobrecapa del *P2P* consisteix en tots els peers que participen com a nodes de la xarxa. Hi ha enllaços entre dos nodes qualsevols que es coneixen, és a dir si un peer participant coneix la localització dun altre peer en la xarxa *P2P*, llavors hi ha un "camí" dirigitdel node anterior al darrer node en la xarxa

de sobrecapa. En base a com els nodes s'enllacen els uns als altres, podem classificar les xarxes del *P2P* com no estructurades o estructurades.

1.3.5 Xarxes *P2P* sense estructura vs. xarxes *P2P* estructurades

La xarxa més exterior o de sobrecapa del *P2P* es basa en tots els peers que participen com a nodes de xarxa. Hi ha enllaços entre dos nodes qualsevols que es coneguin. És a dir que si un peer participant coneix la localització de l'altre peer de la xarxa *P2P*, llavors hi ha un contorn dirigit del node anterior al darrer node en la xarxa. En base a com els nodes en la xarxa s'enllacen els uns als altres, podem classificar les xarxes del *P2P* com:

- **No estructurada:** es forma quan els enllaços de la sobrecapa s'estableixen arbitràriament. Aquestes xarxes poden se construïdes molt fàcilment. El procés s'engega quan un peer vol unir-se a la xarxa, aquests copia els enllaços d'un altre peer per entrar en l'estructura, al cap d'un periode curt de temps crearà els seus propis enllaços. En aquest tipus de xarxa si un peer desitja trobar una dada en la xarxa, la petició té que recórrer tota aquesta amb la finalitat de trobar tants peers com sigui possible que comparteixin aquesta dada. El desavantatge principal d'aquestes xarxes és que les peticions no poden ser resoltes sempre. Un contingut popular és molt probable que estigui disponible en varis peers i qualsevol peer que cerqui el contingut popular, és molt probable que trobi el mateix, però si un peer es troba fent una cerca de dades poc populars que es troben compartides per pocs peers, és molt probable que aquesta cerca no tingui èxit. Com que no hi ha cap correlació entre un peer i el contingut compartit per aquests, no hi ha garantia que la petició trobarà al peer que té les dades desitjades. El *flooding* també causa una alta quantitat de tràfic en la xarxa i per tant aquestes xarxes tenen una eficàcia molt pobre en els resultats de búsqueda.

  Alguns exemples de xarxes *P2P* no estructurades són: *Napster*, *Gnutella* i *KaZaA*.[29]

  En la figura 1.4 es pot veure un esquema d'una xarxa *P2P* de tipus no estructurada.

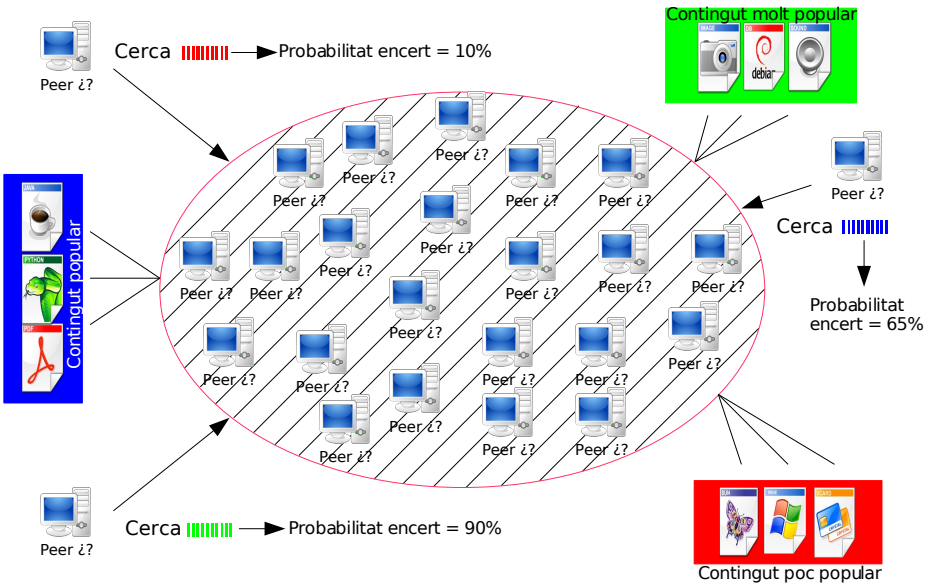

Figura 1.4: Esquema d'una xarxa *P2P* de tipus no estructurada.

- **Estructurada:** superen les limitacions de les xarxes no estructurades mantenint una taula de hash distribuïda (*DHT*) i permitint que cada peer sigui responsable d'una part específica del contingut de la xarxa. Aquestes xarxes utilitzen funcions de hash distribuïdes i assignen valors a cada contingut i a cada peer de la xarxa. Després segueixen un protocol global de determinació de quin peer és responsable de quin contingut. D'aquesta manera, sempre que un peer desitgi cercar certes dades, utilitza el protocol global per determinar el(s) responsable(s) de les dades i després dirigeix la cerca cap al(s) peer(s) responsable(s).

  Alguns exemples de protocols usats en xarxes *P2P* estructurades són: *Chord*, *Pastry P2P Network*, *Tapestry P2P Network*, *Tapestry P2P Network*, *Content Addressable Network* i *Tulip Overlay*.[29]

  En la figura 1.5 es pot veure un esquema d'una xarxa *P2P* de tipus estructurada.

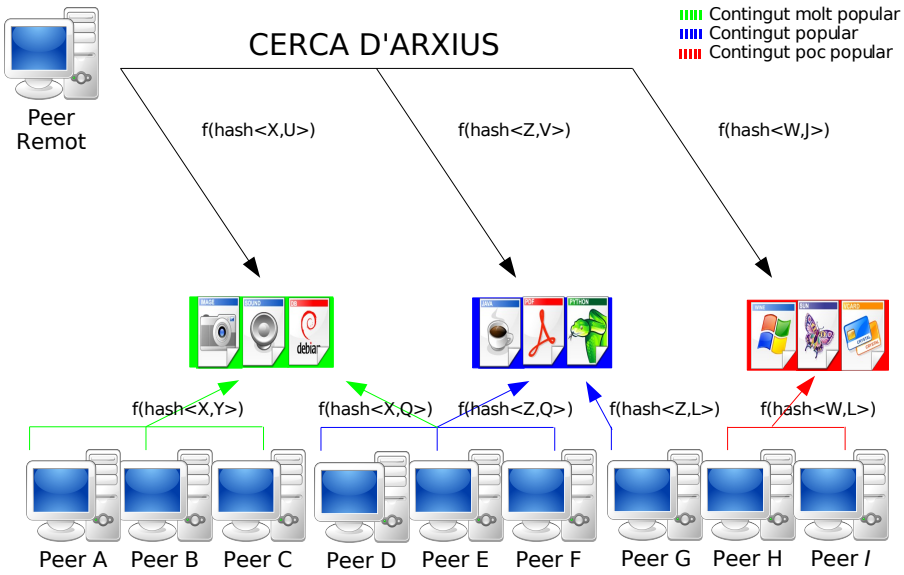

Figura 1.5: Esquema d'una xarxa *P2P* de tipus estructurada.

1.3.6 Ús del *P2P*

Actualment les xarxes *P2P* són molt usades en molts àmbits. Així doncs anem a veure dos exemples d'ús d'aquests tipus de xarxes, que són:

1. **Ús del *P2P* per part de la indústria:** a partir de l'any 2006 companyies americanes i europees fan ús de les xarxes *P2P*. Companyies com *Warner Bros* o la *BBC*, van començar a veure aquestes xarxes com una alternativa a la distribució convencional de pel·lícules i programes de televisió, oferint part dels seus continguts a través de tecnologies com la del *BitTorrent*.

2. **Ús del *P2P* per part dels bioinformàtics:** les xarxes *P2P* han començat a atreure l'atenció de científics de moltes disciplines de la ciència; especialment aquells que gestionen i processen enormes bases de dades, com els bioinformàtics.

   Les xarxes *P2P* poden ésser també usades per fer funcionar sistemes de software dissenyats per realitzar proves que identifiquin la presència de possibles drogues. El primer sistema dissenyat

amb tal propòsit, l'any 2001, en el Centre Computacional per el Descubriment de Drogues (*Center for Computational Drug Discovery*) en la prestigiosa universitat d'Oxford amb la cooperació de la Fundació Nacional per la Investigació del Càncer (*National Foundation for Cancer Research*) d'Estats Units.

Actualment, existeixen varis sistemes de software similars que es desenvolupen en una escala més petita, com sistemes d'administració autònoma pels biòlegs computacionals. Un exemple és el *Chinook*, que s'usa per executar i fer comparacions de dades de caire bioinformàtic amb més dels 25 diferents serveis d'anàlisis que ofereix. Un dels seus propòsits, consisteix en facilitar l'intercanvi de tècniques d'anàlisis dins d'una comunitat local.

Les institucions acadèmiques també han iniciat l'experimentació amb xarxes *P2P* per compartició d'arxius, com és el cas de *LionShare*, que és un projecte que s'ha esforçat per facilitar la compartició legítima d'arxius digitals entre individus e institucions educatives per tot el món.

1.3.7 Reptes del *P2P*

El disseny d'una arquitectura *P2P* de Còmput Distribuït i la gestió que es realitza en la plataforma *P2P* per la distribució de còmput ha de fer front als següents reptes:

- **Gestió distribuida**. Per la compartició de còmput és necessari dissenyar polítiques que basen la seva decisió en informació no completa del sistema i els seus recursos[18, 19]. D'altra banda, tenint en compte que l'escala del sistema pot ésser gran, serà necessari disposar de polítiques que actuin a diferents nivells de gestió: planificador intern del node, planificació d'àrea local i planificació en àrea global.

- **Tolerància a errades**. El gran dinamisme que caracteritza un sistema *P2P* (contínues entrades i sortides de nodes sense previ avís) i el gran nombre de components que poden formar part d'aquests, fa que la possibilitat d'errada d'un node sigui molt elevada. L'execució distribuida de les aplicacions s'ha de portar a terme de forma transparent als fallos que sorgeixin, de tal manera que el sistema garantitzi el resultat final independentment d'aquesta causística. Per això, serà necessari la definició de mecanismes de reconfiguració de l'aplicació per llençar nòvament el còmput perdut i mecanismes de reconfiguració de l'arquitectura per mantenir l'estat del sistema des del punt de vista de recursos i tasques[20, 16].

- **Auto-organització**. En els sistemes P2P els rols que tenen que "jugar" cada un dels nodes no estan definits a priori. Aquests papers com a proveïdors de recursos, gestors o planificadors es van assignant de forma dinàmica en funció de les necessitats del sistema. En el moment que un node amb responsabilitats de gestió cau, el sistema ha de garantitzar que un altre node assumirà aquesta gestió.

- **Gestió de recursos heterogenis**. La gestió de recursos en una plataforma P2P introdueix dos reptes importants: l' heterogeneïtat dels recursos i la gestió de la compartició. L' heterogeneïtat dels recursos pot afectar a l'execució de les aplicacions i les prestacions oferides per l'arquitectura. Per tant, és crític el disseny de polítiques distribuides i dinàmiques de planificació i balanceig de càrrega que permetin emmascarar aquesta heterogeneïtat, per proporcionar una gestió eficient dels recursos[21, 22, 23]. D'altra banda el principi bàsic sobre el que es fonamenten els sistemes P2P és la compartició de recursos. Per que aquesta es porti a terme de forma justa i proporcional als

recursos oferits al sistema es necessita comptabilitzar la utilització/cessió de recursos per part dels usuaris del sistema i fomentarla mitjançant polítiques per incentivar la compartició[24].

- **Seguretat**. Els nodes d'una xarxa P2P tenen que interactuar amb altres nodes que són desconeguts i per tant és necessari gestionar el risc derivat d'aquestes interaccions (transaccions), sense comptar amb la presència d'una tercera entitat/autor de confiança. Per superar aquests problema, és necessita usar tècniques d'encriptació, autenticació i execució de codi en entorns segurs[25, 26, 27].

Al mateix temps, s'ha de fer front als reptes que planteja el disseny d'una arquitectura P2P en l'entorn de les aplicacions actuals, és també important garantitzar un temps d'execució i cost acotats. Per això s'ha de dissenyar un sistema on l'usuari de les aplicacions puguin especificar els següents requisits de qualitat de servei respecte a la seva execució: temps de resposta i retorn de l'aplicació, cost d'execució i disponibilitat del servei.

1.4 JXTA

JXTA[6, 6, 7], que ve de la paraula Juxtapose, és una plataforma peer-to-peer de codi obert creada per Sun Mycrosystems al febrer de 2001 amb la col·laboració d'investigadors de vàries institucions acadèmiques. Aquesta plataforma defineix un conjunt de protocols basats en XML que permet a qualsevol tipus de dispositiu connectat a la xarxa intercanviar missatges i col·laborar amb independència de la topologia de la xarxa. JXTA és el *framework* per xarxes de peers més madur en l'actualitat i va ser dissenyat per suportar que un gran ventall de dispositius - PCs, servidors, telèfons movils, PDAs- es comuniquessin d'una forma descentralitzada.

Com JXTA està basat en un conjunt de protocols oberts, pot ser portat a qualsevol llenguatge de programació modern. La versió Java és la més madura de totes i també hi ha un versió en C/C++, JXTA C/C++, que també està sent desenvolupada.

Els peers de JXTA creen un entorn de xarxa virtual que permet a un peer interactuar amb altres peers directament fins i tot quan aquests estan darrera de firewalls i enrutadors o utilitzen diferents protocols de transport. A més, cada recurs s'identifica per un únic ID, 160 bits SHA-1 URN. Això implica que un peer pot canviar la seva ubicació i mantenir el seu identificador.

A més dels identificadors, JXTA proveix d'altres funcionalitats bàsiques com el descobriment de peers, els grups de peers, i els pipes.

1.4.1 Conceptes principals

En aquest apartat s'exposen els conceptes principals que dona JXTA per crear una xarxa de peers.

1.4.1.1 Endpoints

Un endpoint es pot definir com l'emisor o receptor de la font de qualsevol conjunt de dades que són transmesses sobre la xarxa de peers. Un endpoint proporciona abstracció sobre les interfícies de la xarxa usades per a enviar i rebre dades oblidant-se de detalls com protocols inferiors o arquitectura del sistema.

1.4.1.2 Advertisement

Un *advertisement* és una manera de presentar a la resta de la xarxa els recursos disponibles i la seva ubicació. Els anuncis permeten simplificar la tasca d'organitzar les xarxes de peers.

Formalment, un anunci es defineix com una representació estructurada d'una entitat, servei o recurs disponible per part d'un peer o un grup de peers com part de la xarxa P2P.

Tots els recursos utilitzats per JXTA poden ser representats com un anunci, incloent-hi peers, grups de peers, pipes, endpoints, serveis... etc.

1.4.1.3 Pipe

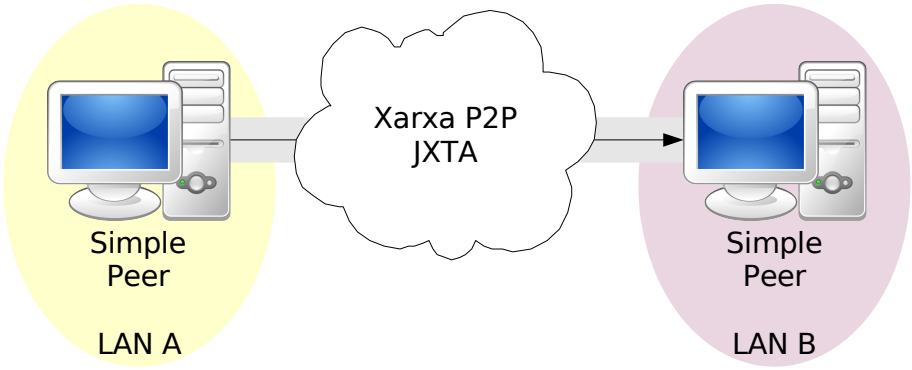

Figura 1.6: Funcionament d'un pipe en JXTA.

Un pipe es defineix com un objecte d'un sistema operatiu utilitzat per connectar la sortida d'un procés amb l'entrada d'algun altre. Amb JXTA, un pipe és un canal virtual de comunicacions unidirecional i asíncron, que connecta dos o més endpoints. Els pipes poden disposar d'altres mecanismes addicionals com el xifrat.

Els pipes constitueixen l'eina principal per la comunicació d'una xarxa peer-to-peer implementada amb JXTA.

La figura 1.6 mostra les possibilitats que dona JXTA per establir comunicacions. Gràcies a JXTA el programador pot oblidar-se d'identificar un peer o tenir en compte la xarxa en la que es troba, establint únicament els endpoints (els peers que volen establir la comunicació). El sistema propagarà els missatges i envia els missatges tenint en compte els routers, firewalls i demés obstacles en la comunicació.

1.4.1.4 Peers

Un *peer* és un node dins una xarxa P2P que forma la unitat de processament fonamental en qualsevol solució peer-to-peer. Fins ara s'havia de definir un peer com una aplicació funcionant en un sol ordinador connectat a una xarxa com Internet, però aquesta definició limitada no inclou la veritable funció que té un peer, ja que descarta la possibilitat de que un peer pugui ser una aplicació distribuïda entre diferents màquines o que pugui ser un petit dispositiu, com un PDA, el qual es connecta a una xarxa indirectament. I des d'aquest punt de vista, una sola màquina pot suportar múltiples peers al mateix temps.

Per englobar tot això, en aquest treball, es definirà un peer com:

"Qualsevol entitat capaç de realitzar un cert treball útil *i de comunicar els resultats, directa o indirectament, a una altra entitat sobre una xarxa."*

La definició de *treball útil* depèn del tipus de peer que sigui. Existeixen tres tipus de peers en una xarxa P2P de JXTA:

1. **Simple Peers**: Un simple peer està dissenyat per servir un sol usuari, habilitant a aquest la possibilitat de proveïr serveis del seu dispositiu i consumir-ne de proveïts per altres peers de la xarxa. Com a norma general, un simple peer està situat darrera d'un tallafocs, separat de la xarxa. Els peers fora del tallafocs probablement no seràn capaços de comunicar-se directament amb un Simple peer situat darrera el tallafocs. Degut a la seva accessibilitat limitada a la xarxa, els single peers tenen la responsabilitat més baixa en les xarxes P2P JXTA.

2. **Rendezvous Peers**: En general, un rendezvous és una porta o un punt de trobada; en P2P, un peer rendezvous proveeix els altres peers amb una localització de la xarxa usats per descobrir altres peers i recursos d'aquests. El rendezvous peer emmagatzema la informació dels altres peers i la posa a disposició de la resta de la xarxa.

   Un peer rendezvous pot augmentar la seva capacitat dipositant la informació sobre els peers per un ús futur. Aquest tipus de peers tenen el potencial de millorar el tallafocs, de reduir tràfic de la xarxa, i de proporcionar un servei millor als simple peers.

   Com es pot veure a la figura 1.7 un peer rendezvous és l'encarregat de propagar els missatges de la xarxa local a altres equips de fora. Això permet que amb un únic peer amb sortida a l'exterior, la resta també es puguin comunicar.

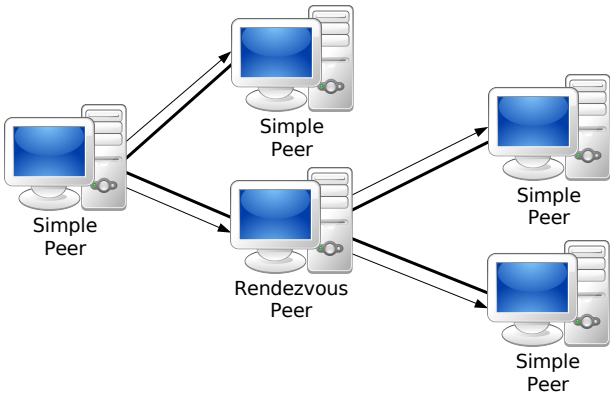

Figura 1.7: Rendezvous Peer propagant missatges.

   Un peer rendezvous existirà generalment davant d'un tallafocs de xarxa, tot i que podría existir darrera, però hauria de ser capaç d'atravessar-lo utilitzant un protocol autoritzat pel tallafocs o un router peer de fora.

3. **Router Peers**: Un router peer (o relay peer) proveeix d'un mecanisme als altres peers de la xarxa per que puguin comunicar-se amb altres de fora de la xarxa separats per un tallafocs o per un equipament que utilitzi NAT (Network Address Translation).

   Per enviar un missatge a un peer per mitjà d'un router, qui envia el missatge ha de determinar quin router utilitzarà per comunicar-se amb el peer destí. Aquesta informació d'encaminament proporciona un mecanisme al sistema P2P per a substituïr el DNS tradicional i que permet a un dispositiu connectat dinàmicament amb una adreça IP dinàmica ser trobat a la xarxa.

   A la figura 1.8 es mostra com mitjançant un router peer (o rely peer) es pot aconseguir que tota una xarxa es pugui comunicar amb l'exterior.

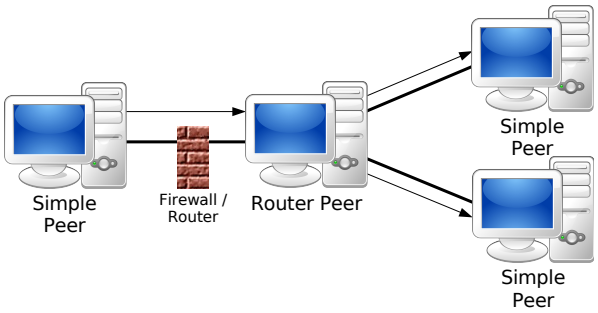

Figura 1.8: Router Peer enviant missatges a través d'un firewall.

1.4.1.5 Peer Groups

Abans de JXTA, la naturalesa de les solucions P2P i dels seus protocols associats consistia en dividir l'espai de la xarxa segons l'ús. Si es volia realitzar algun tipus de compartiment d'arxius, s'utilitzava el protocol de Gnutella que simplement permetia la comunicació amb altres peers que utilitzessin el mateix tipus de protocol. De manera similar, si el que es volia era realitzar missatgeria instantània, s'utilitzava ICQ i es podia comunicar solament amb altres peers que també usaven ICQ.

Les incompatibilitats dels protocols redueixen l'eficàcia d'ús de la xarxa per part dels peers. Si es considera un sistema P2P en el qual tots els clients entenen el mateix sistema de protocols, com passa amb JXTA, el concepte d'un peer group és necessari per subdividir l'espai de la xarxa. Per tant, es defineix un *peer group* com:

"Un sistema de peers format per a servir un interés comú o una fita dictada pels peer implicats. Els peer groups poden proporcionar serveis als seus peer inaccessibles per altres peers en la xarxa P2P."

Els diferents motius pels quals la palataforma JXTA divideix la xarxa de peers en grups són:

- **L'aplicació en la que han de col·laborar com un grup.** Un peer group està format per donar un servei d'intercanvi als membres que no desitgen tenir disponibilitat per tota la xarxa P2P. Una raó de fer això podia ser la privacitat de les dades usades per l'aplicació.
- **Els requisits de seguretat dels peers implicats.** Un peer group pot emprar serveis d'autenticació per a restringir l'accés al grup i els serveis oferts per aquest.
- **La necessitat de la informació d'estat sobre els membres del grup.** Els membres d'un peer group poden supervisar a altres membres. La informació d'estat ajuda a mantenir un nivell mínim del servei per a l'aplicació del peer group en qüestió.

Els membres del peer group poden proporcionar accés redundant a un servei, assegurant-ne així que sempre estigui disponible.

1.4.2 El projecte JNGI

Actualment no hi han massa plataformes P2P de computació distribuïda. A continuació es descriu un projecte desenvolupat amb aquesta filosofia, *JNGI* [8]

JNGI és una plataforma de computació distribuïda creada utilitzant les llibreries P2P JXTA[4]. Aquestes extensions atorguen a altres desenvolupadors la possibilitat de fer aplicacions P2P complexes d'una manera més fàcil. Aquest projecte va ser proposat al 2002 amb el titol "Framework for Peer-to-Peer Distribution Computing in a Heterogeneous Decentralized Environment", és a dir, un marc de treball per a la computació distribuïda P2P per entorns descentralitzats i heterogenis. En aquest projecte es descriu un marc de treball per una xarxa de computació distribuïda JXTA que permet ser utilitzat per resoldre problemes CPU-intensius. La fita de JNGI es permetre a un desenvolupador amb poc coneixement de JXTA elaborar un programa amb tot el potencial possible dels peers JXTA.

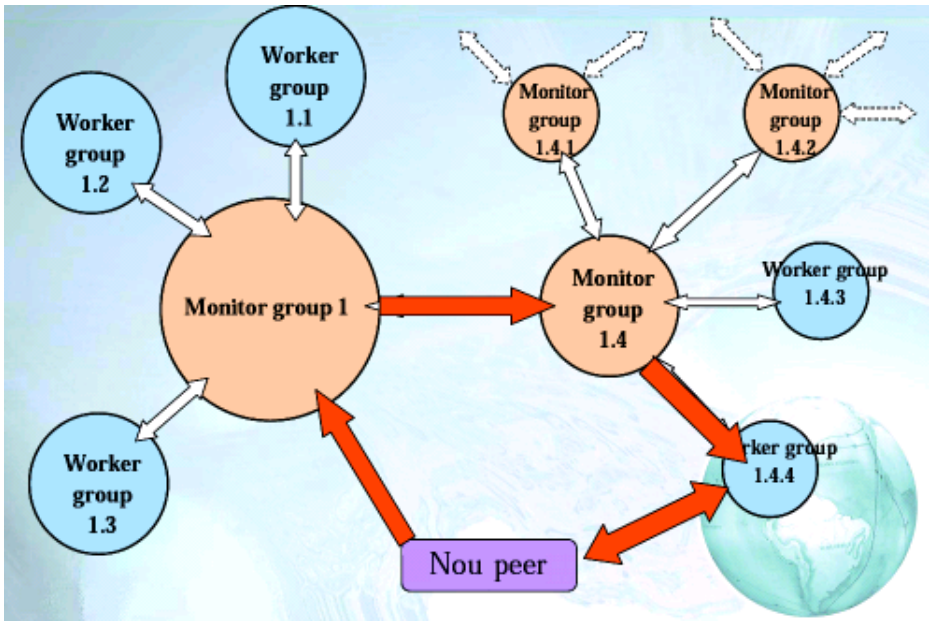

Figura 1.9: Distribució dels peers a JNGI.

JNGI defineix un *Monitor*, un *Worker* i un *Task Dispatcher peer group*[5]. Els membres d'aquest grup de peers són utilitzats per a definir els rols dels altres peers participants. Un peer pot ser un membre d'un o de varis grups i pot haver una o més instàncies en cada grup de peers en una xarxa JNGI. El *Peer Group Monitor* es qui decideix sobre les peticions per entrar en la xarxa JNGI. El grup monitor també determina quins grups i quins rols s'han d'assignar a cada peer.

El *Peer Group Monitor* atorga un membre subministrador de tasques a la xarxa, com qualsevol altre peer. Un cop atorgat el membre, el subministrador de tasques subministra una nova tasca consistent en les dades i el programa. El programa agafa la forma d'una classe Java, implementant java.lang.Runnable i java.lang.Serializable, compilat amb el codi byte[6] estàndard. Tots els peers *Worker* involucrats en la mateixa tasca han d'executar la mateixa classe Java. L'única diferència són les dades d'entrada. Aquestes agafen la forma d'instàncies serialitzades de classes Java. JNGI es refereix a aquestes instàncies com tasques. Aquestes tasques són instanciades amb dades diferents abans de ser serialitzades i distribuïdes.

El grup de peers de distribució de tasques reb el codi byte i les tasques del subministrador de feina.

[4]Veure secció 1.4 JXTA.
[5]Grup de peers de distribució de tasques.
[6]El bytecode és un codi intermedi més abstracte que el codi màquina.

Els peers *Worker* sol·liciten feina constantment al distribuïdor de tasques des que entren a formar part de la xarxa JNGI. Un cop que el distribuïdor té un joc de tasques per distribuir, respon als *Workers* amb el codi byte, si és necessari, i la tasca. El *Worker* utilitza el codi byte per deserialitzar la tasca i poder tractar l'objecte com un Thread normal de Java. L'objecte completa la seva feina i posa el resultat en una variable. Posteriorment és retornat asíncronament al grup distribuïdor de tasques. D'altra banda, el *Task Dispatcher group* té la capacitat d'enviar la mateixa tasca a múltiples *Workers*. En aquest cas, el *Task Dispatcher group* utilitzaria el primer resultat rebut i descartaria la resta.

Des de que se l'hi assigna un treball, de forma constant se li sol·liciten al distribuïdors tasques finalitzades. Un cop aquest últim rep la resposta de cada tasca, les retorna al sotmetedor del treball amb tots els objectes serialitzats que li hagin retornat els *Workers*. Per acabar, cal remarcar també que l'assignant del treball és responsable de qualsevol procés que pugui ser necessari.

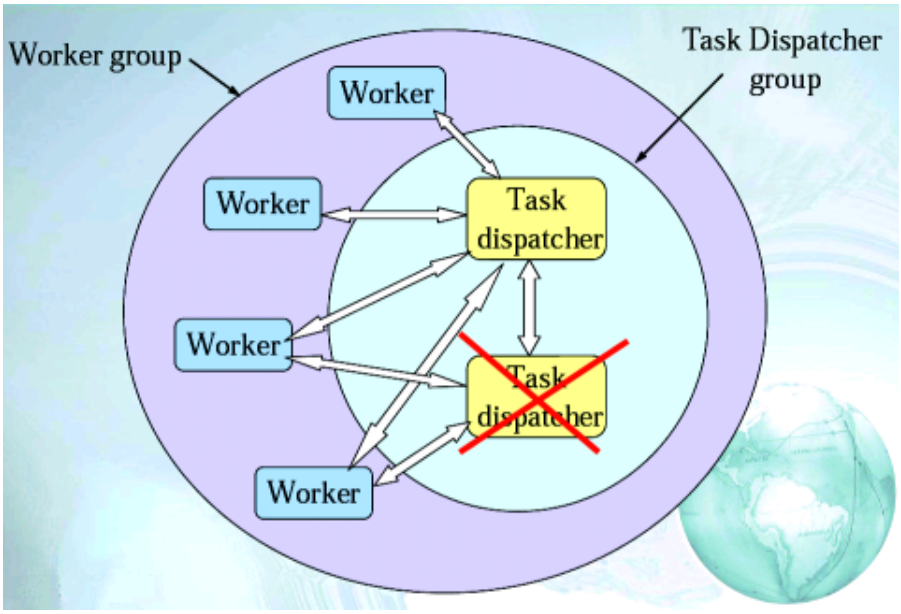

Figura 1.10: Distribució del grups a JNGI.

1.5 Objectius del treball

A grans trets, l'objectiu d'aquest treball és desenvolupar una interfície gràfica del sistema de computació distribuïda peer to peer *CompP2P*. Aquest projecte estarà ben documentat, per tal que les millores futures es puguin introduïr amb facilitat. També s'intentarà seguir les màximes de flexibilitat, robustesa, eficiència, eficàcia i diversitat que el projecte primitiu tenia.

D'altra banda, també s'efectuaran algunes petites i puntuals modificacions del projecte anterior per tal d'efectuar una millor integració del sistema inicial amb la seva interfície gràfica.

Per tant el repte principal és el de dissenyar i implementar una interfície gràfica que permeti l'ús del sistema de còmput distribuït amb arquitectura *P2P*.

Un punt fort del sistema *P2P* i per tant de la seva interfície gràfica, és el fet que treballarà sobre plataformes diferents. Per tant l'abast del projecte ha d'ésser global, per això és interessant desenvolupar

l'arquitectura del projecte amb *Java Swing*[12] per poder fer una aplicació gràfica de sistema de peers el més gran i heterogènia possible; tot i que s'ha posat especial cura amb els sistemes operatius basats amb *UNIX*[7]. Dins d'aquest punt inclourem el fet que s'ha de tractar d'una aplicació de fàcil instal·lació i gestió per facilitar la seva usabilitat entre els usuaris de tot el món.

A continuació es detallen els objectius específics que el disseny i la implementació d'aquesta aplicació han de complir.

1. El nombre d'interfícies gràfiques realitzades i relacionades amb sistemes *P2P* són moltes, per tant podem adoptar idees de disseny ja establertes fixant-nos en els dissenys ja efectuats; per exemple el disseny gràfic de l'aplicatiu, ha estat inspirat amb la interfície gràfica del programa d'intercanvi *Amule* (tal i com es pot veure en la figura 1.11). Tot i això, les inspiracions han estat purament gràfiques, ja que no s'ha efectuat cap estudi del codi d'aquesta (ni d'altres) aplicacions del gènere per tal d'evitar corrompre el codi del propi projecte.

2. Tenint en compte que aquest projecte té la intenció de seguir sent desenvolupat i millorat, el disseny ha de ser modular i fiable, permetent una fàcil modificació i/o ampliació.

3. La capacitat de càlcul del sistema tendeix a ser directament proporcional al número d'usuaris d'aquest, i per tant, quant més usuaris hi hagi major serà aquesta capacitat de còmput. És per això que sorgeix la necessitat d'arribar al màxim d'usuaris possibles. Això es pot aconseguir portant a terme una aplicació que es pugui executar en molts sistemes diferents, per tant es desenvoluparà una aplicació multiplataforma.

4. Per aconseguir l'independència del sistema del punt 3 s'ha escollit el llenguatge de programació *Java* usant l'eina *Swing (Standard Window Toolkit)*. Aquesta eina sorgeix de l'evolució de *AWT (Abstract Window Toolkit)*. És una eina que té com a finalitat la creació d'interfícies gràfiques multiplataforma, configurable i adaptable, fet que ens permetrà assolir els objectius del nostre treball.

5. Caldrà integrar-se amb el sistema inicial *CompP2P* per tal de connectar la interfície gràfica amb el sistema base i també per efectuar alguna millora del sistema base en aspectes que siguin connexos a la interfície gràfica d'aquest.

6. La redacció d'un manual d'usuari de l'aplicació per tal que qualsevol persona conegui el 100% de les possibilitats que té l'aplicació, així com els passos a seguir en cas que es vulgui millorar aquesta.

Un cop desenvolupada tota l'aplicació cal comprovar el funcionament del programa en un entorn real, com pot ser un laboratori, i posar-lo a prova per veure la seva funcionalitat i el seu rendiment real.

Tots aquests punts tenen com a objectiu final, la creació d'un entorn de treball amigable i fiable d'una aplicació que suposa una forta innovació en el camp de la computació distribuïda, facilitant que "més ulls" dirigeixin la seva atenció cap a aquesta nova àrea de recerca amb grans possibilitats de futur.

1.6 Enfocament i mètode seguit

El mètode de treball ha estat enfocat a la resolució de petits problemes per tal d'afrontar amb garanties el projecte. S'anaven dissenyant diferents àrees de l'aplicació i es testejaven per veure el correcte funcionament.

[7] Com són Linux o Mac Os X Tiger.

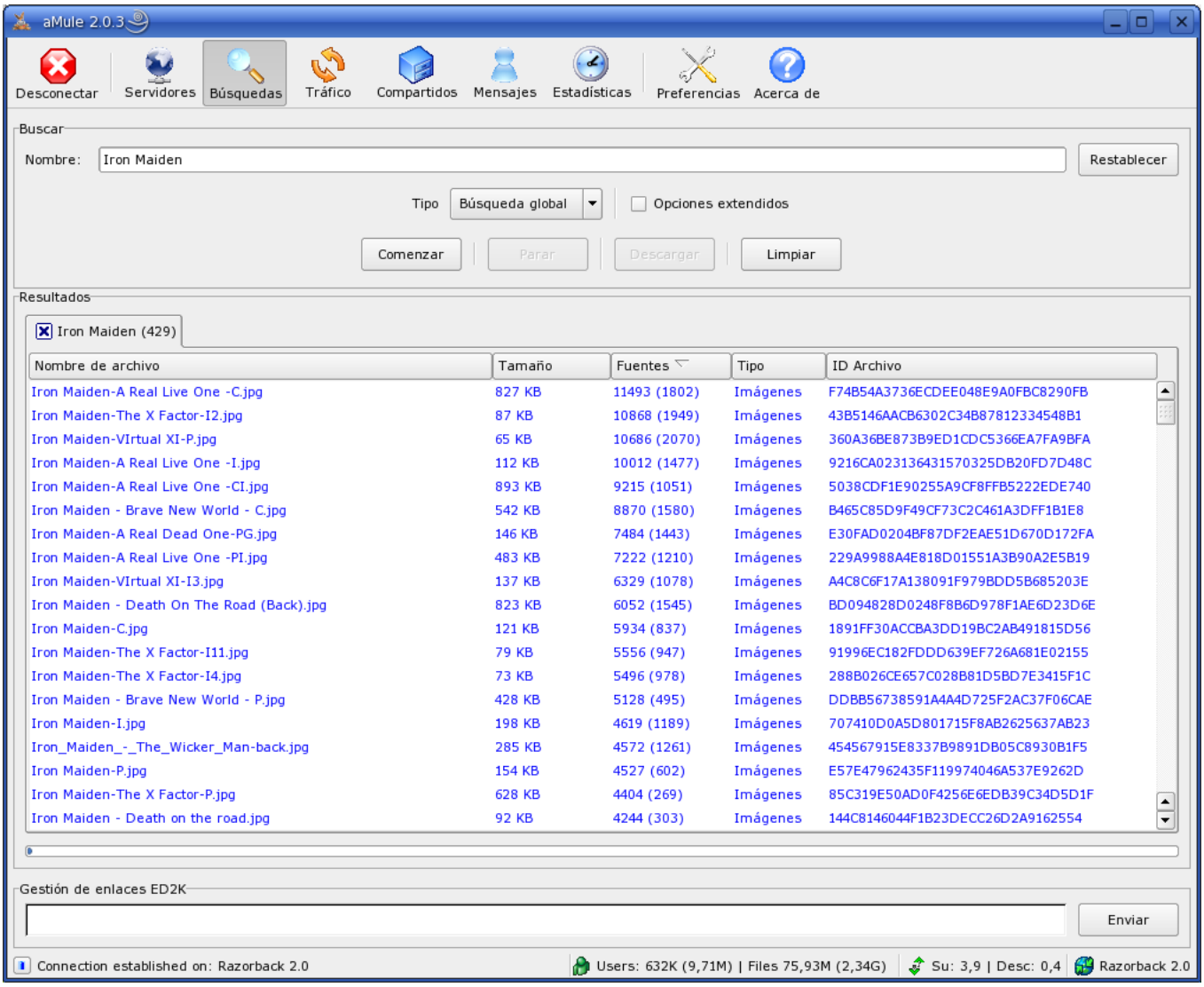

Figura 1.11: Captura de l'aplicació *Amule*; font d'inspiració gràfica del projecte *CompP2P-GUI*.

Aquest mètode de disseny de *software*, és el recomanat per tal d'anar resolent els reptes que sorgeixen en el desenvolupament de tot el projecte.

A meitat del projecte s'ha efectuat una reestructuració de mètode de treball, s'han dividit el nombre de fitxers segons temàtica i tasques efectuades i s'han testejat cada una d'elles.

Aquesta estructura més organitzada i coherent ha facilitat la tasca del desenvolupador. A més a més, aquesta organització, permetrà efectuar posteriors millores de forma senzilla per part de persones externes al projecte.

Finalment, unn cop dissenyades totes les àrees de l'aplicació, s'ha efectuat un repàs general de totes i cadascuna d'elles per testejar la seva harmonia i funcionalitat amb tot el sistema, així com la compenetració del sistema primitiu *CompP2P*.

Per complir l'objectiu d'un disseny estructurat, s'ha escollit un disseny de l'aplicació en mòduls independents. Amb això es pot implementar un mòdul per complet, sense necessitat de tenir implementat cap altre mòdul. La metodologia que s'ha seguit en totes les fases de desenvolupament de l'aplicació ha estat totalment orientada a objectes. S'ha intentat aconseguir un sistema fortament cohesionat.

Pel que fa a les eines de treball, per la implementació s'han utilitzat dos editors de text pla (*Kate* i *Smultron*) segons es programava en l'entorn *Linux* o *Mac Os X*. El compilador utilitzat ha estat el de *java sun* en la versió 5.0.

Quant a la documentació s'ha utilitzat l'eina nativa de *java*, *javadoc*, per dur a terme la *API* del programa. I pel que fa a la redacció d'aquesta memòria s'ha utilitzat LaTeX fent ús del *framework Kile*.

1.7 Contingut

En aquest apartat es fa una breu descripció dels altres capítols de la memòria. La relació de capítols i seccions d'aquest treball és la següent:

- **Capítol 2: *CompP2P***. És una breu explicació de l'arquitectura que constitueix el sistema primitiu *CompP2P*.
- **Capítol 3: *CompP2P-GUI***. És el cos del treball i un dels capítols més importants; en aquests s'explica detalladament com és l'aplicació en si, és a dir, s'explicarà tot el disseny, això és, especificació i implementació de totes les classes dels diferents mòduls així com una pinzellada de les classes més importants.
- **Capítol 4: Manual *CompP2P-GUI***. Conté el manual d'usuari de l'aplicació. S'explica pas a pas i mitjançant un exemple pràctic, la manera d'adaptar el disseny d'una aplicació distribuïda per al sistema *CompP2P*, així com la metodologia a seguir per duur a terme modificacions en el *GUI*.
- **Capítol 5: Adaptar aplicacions al model *CompP2P***. Conté una explicació de com adaptar una aplicació qualsevol a una que es pugui fer funcionar en el sistema *CompP2P*.
- **Capítol 6: Conclusions**. S'exposen les conclusions extretes de la realització del treball.
- **Capítol 7: Treball Futur**. Es llisten una sèrie de propostes de treballs futurs per millorar l'aplicació desenvolupada en aquest treball.
- **Apèndix A: Codi CompP2PGUI**: Conté el codi de la classe principal *CompP2PGUI*.

Capítol 2

CompP2P

CompP2P és una aplicació desenvolupada amb l'objectiu d'executar treballs remotament entre peers. El seu desenvolupament ha estat dut a terme en Java utilitzant la plataforma JXTA per intercomunicar els diferents participants en l'aplicació. S'executa com un dimoni del sistema que espera rebre treballs distribuïts per ser executats en un entorn P2P.

Per la comunicació entre els diferents peers del sistema s'utilitza JXTA. Per la comunicació amb l'usuari utilitza sockets TCP.

2.1 Arquitectura

En aquesta secció s'explica el funcionament global del sistema de còmput distribuït entre iguals, *CompP2P*. Tenint en compte els objectius marcats, es raonen i es presenten les solucions que implementa aquesta plataforma.

El sistema necessita una gran escalabilitat i poder estar format per un gran número d'usuaris. Això implica que la comunicació entre els peers ha de ser versàtil i dinàmica. D'altra banda, això implica un gran nombre de comunicacions i per tant un baix rendiment del sistema. Per aquest motiu, s'introdueix una jerarquia dels equips a la xarxa.

En els següents apartats, es donarà una visió detallada de cadascun dels aspectes fonamentals de l'arquitectura que utilitza *CompP2P* i la motivació d'aquestes solucions.

A la figura 2.1 es mostra un esquema on es pot apreciar l'arquitectura de l'aplicació. Al llarg del capítol s'explica detalladament cadascuna de les parts que composen els diferents nivells d'aquesta arquitectura.

2.1.1 Mode d'execució

CompP2P ha de tenir una funcionalitat clàssica d'un servidor. És a dir, ha d'estar esperant a rebre peticions de feina permanentment, en aquest cas des d'una plataforma *peer to peer*. D'aquí neix la necessitat de que la base del sistema s'executi com un dimoni, esperant rebre treballs per executar.

Però si es té un dimoni, com es pot fer per executar un treball al sistema? La solució més simple consisteix en crear una aplicació que utilitzi JXTA, per tal que aquesta es connecti a qualsevol peer i finalment enviï la feina que es pretén executar o la informació que es vol aconseguir. Tot i ser una solució bastant bona, ha estat descartada ja que viola la filosofia de l'arquitectura de que tots els usuaris són iguals. Es podria donar el cas de tenir 50 usuaris que volen executar treballs i un únic executador de codi, això seria definitivament una arquitectura client-servidor.

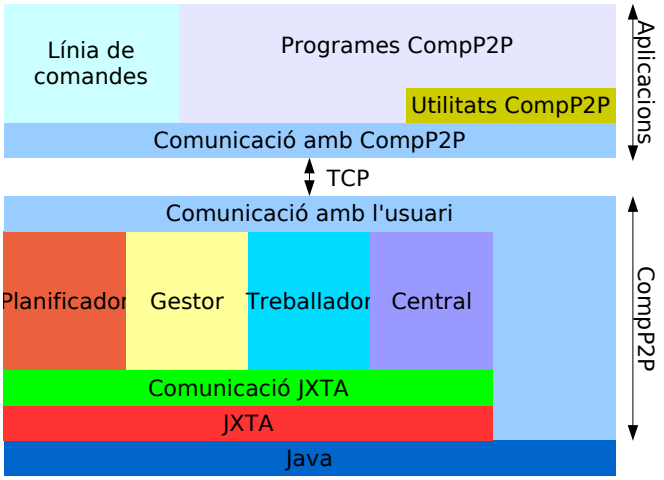

Figura 2.1: Esquema de l'arquitectura de CompP2P.

L'altra possible solució és dotar el dimoni d'una forma de comunicació externa. La forma més senzilla és utilitzar sockets i tenint en compte el caràcter multiplataforma de *CompP2P* i la necessitat de rebre les dades correctament i en ordre, fan decantar la balança pels sockets TCP.

Vist això, el dimoni *CompP2P* ha de tenir un mòdul de comunicació extern per TCP, on rebrà les peticions d'execució de treballs i sol·licituds d'estadístiques. El port TCP de comunicació elegit ha estat el 35557, sense cap raó de pes, tret de que és un port sense ús conegut i lliure.

Per tant, el dimoni de *CompP2P* es comunica amb altres peers mitjançant la plataforma JXTA i es comunica amb l'usuari per TCP.

A més, l'aplicació ha de ser totalment multiplataforma, i per tant, cada plataforma, com a tal, té les seves particularitats. Una de les característiques identificatives més freqüents és el caràcter de separació d'arxius, així, per exemple als sistemes UNIX s'utilitza "/" i al sistema Windows "\". Per tant, a l'hora de crear els directoris, cal utilitzar la crida *System.getProperty*.

2.1.1.1 Threads

Un dels fonaments de *CompP2P* és l'utilització de fills d'execució. Un *thread* és la unitat d'execució més petita del kernel i com a mínim existeix un *thread* en cada procés. En aquest cas, s'utilitza més d'un fil i tots aquests compartiran la mateixa memòria i recursos.

Un *thread* és una manera d'executar dues o més tasques simultànies. Múltiples *threads* poden ser executats en paral·lel, *multithreading* i generalment es produeix quan un processador intercanvia entre diferents *threads*. En el cas de Java s'utiltiza el suport que dona la màquina virtual per mitjà de la classe Thread i la interficie Runnable.

CompP2P ha de dur a terme més d'una tasca a la vegada, com pot ser rebre missatges de l'usuari, controlar l'estat del sistema peer to peer o executar treballs. És per això que l'ús de fils d'execució és un dels seus punts claus.

La majoria dels fils són autonoms i la seva comunicació és entre màquines remotes via JXTA i es fa per enviament de missatges. Un dels mecanismes que s'utilitza per sincronitzar els *threads* locals són els *semàfors*. Un dels exemples més significatius és el d'una tasca que s'ha retornat al peer *master* i s'ha d'informar al fil que està a l'espera d'aquest fet. Amb això s'evita realitzar esperes actives.

2.1.2 Grups

Com ja s'ha plantejat amb anterioritat, *CompP2P* necessita d'una segmentació i una jerarquia dinàmica de la xarxa. En aquest apartat, es mostra en detall les solucions aplicades.

Per dotar el sistema d'una comunicació global, tots els peers formaran part del mateix grup, el grup de treballadors. Cada peer haurà de conéixer l'estat dels altres per l'execució remota. Si el grup està format per un miler de peers, al sistema global hi haurà un milió d'entrades redundants i per tant un milió de comunicacions. Això implica un gran problema d'ineficiència de recursos.

Per solucionar aquest problema, introduirem el concepte de *manager*. Únicament aquest tipus de peer coneixerà l'estat de la resta. A més de la redundància d'informació, amb això reduirem el número de comunicacions.

En aquest punt esdevé un segon problema, si tenim un únic peer gestor, les comunicacions poden tenir un retard massa gran per les necessitats del sistema. Per exemple, si el gestor està separat dels altres peers per tres encaminadors, les comunicacions no seran òptimes. Per solucionar aquest problema i reduir la càrrega del manager, segmentarem la xarxa en grups d'àrea.

Per cada àrea es designarà un *gestor*, amb això s'aconsegueix optimitzar les comunicacions a més de reduir la càrrega i la saturació de recursos d'un suposat únic gestor. Els diferents gestors, han de poder comunicar-se entre si, per poder fer un sistema de còmput global i no moltes petites àrees. Per aquest motiu, tots el peers hauran de pertànyer a un grup germà del global, on comunicar-se entre ells i poder coneixer-se fàcilemnt, aquest serà el grup de monitors.

Ara cal definir el concepte de *grup d'àrea*. Tenint en compte que la característica principal és que la velocitat de les comunicacions sigui alta, podem aprofitar que si el grup pare no té servei rendezvous, els missatges dels subgrups no poden travessar encaminadors. Tenint en compte això, si el grup de treballadors no propaga l'existència del grup inferior cap a fora del router, tindrem la xarxa separada en àrees delimitades pels encaminadors.

La distribució dels grups de peers de *CompP2P* serà la que s'observa a la figura 2.2, un grup de treballadors on estan tots els peers, segmentat en arees amb un dels peers amb el rol de gestor del grup d'àrea i a la seva vegada els gestors, formen part d'un grup de monitors d'àrea.

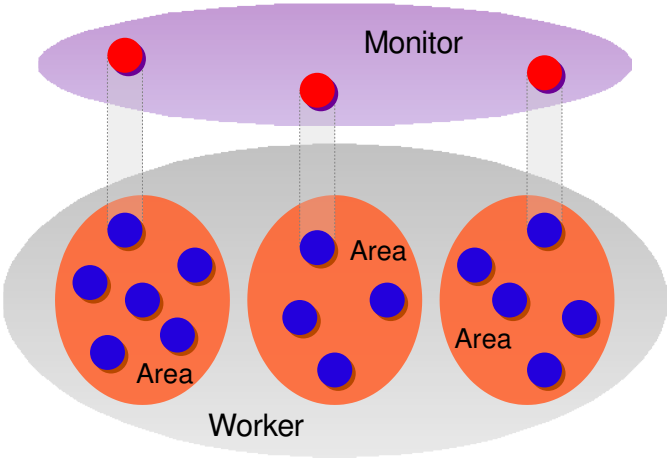

Figura 2.2: Jeararquia de grups de CompP2P.

Tot i això, a vegades que la creació de grups d'àrea sigui dinàmica pot ser un inconvenient. Per

exemple, si volem utilitzar tota una sala d'ordinadors per que treballi conjuntament, pot donar-se el cas, que el servei Rendezvous no trobi en aquell moment el grup o bé el sitema no vegi que allò és potencialment un àrea, i es crein petits grups.

Per donar més possibilitats i solventar aquesta problematica, s'ha introduït la figura de la creació estàtica de grups. Per fer això, és guardarà un avís del grup que es vulgui crear estàticament en un arxiu amb el nom del grup. Si el programa troba aquest avís de grup, es conectarà a aquell grup de peers. Per exemple, si volem que tota una aula estigui sempre al mateix grup, crearem l'arxiu "CompP2Parea.adv" amb l'identificador de grup que es vulgui utilitzar, i aquest es guardarà a tots els equips que es vulgui que formin part.

2.1.2.1 Situacions dels grups

Un cop ja s'ha definit com es distribuiran els diferents grups de peers, cal veure com funciona la creació i el manteniment de la jerarquia i el rol que prendrà i en quin moment dins d'aquesta.

A la figura 2.3 es mostra un diagrama d'estats que mostra el procediment que segueix i els rols que pren un peer quan entra en el sistema de peers.

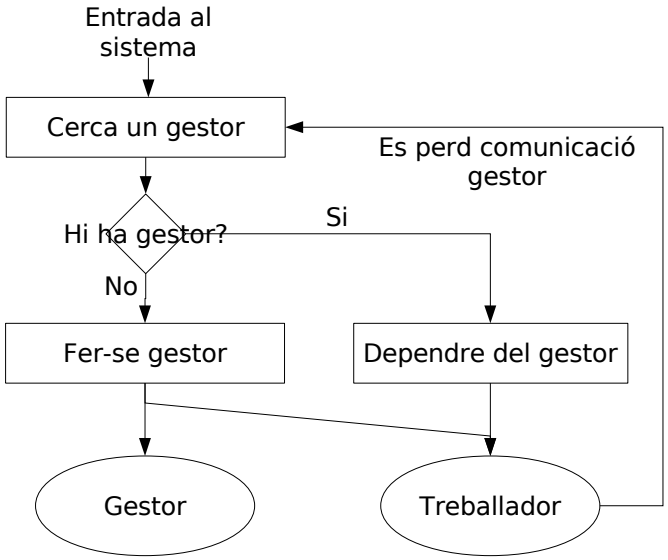

Figura 2.3: Diagrama de l'entrada al sistema CompP2P.

A continuació, es detallarà els processos d'entrada, manteniment i sortida de *CompP2P*.

Entrada al sistema Per estudiar la creació de la jerarquia i els rols que va prenent cada peer en cada situació, es poden donar tres casos diferents:

- Partint d'un sistema sense cap usuari, quan un peer vol formar part de la plataforma de computació p2p, segueix els següents passos:

  1. Cerca el grup de treballadors i com encara no s'ha iniciat cap cop i per tant no hi ha cap grup creat, el crea i en passa a formar-ne part.
  2. Cerca el grup d'àrea i igual que abans el crea i passar a formar-ne part.

3. Cerca d'un manager dins del grup d'àrea i com no n'hi ha, crea un manager i es connecta a ell. En aquest cas, el gestor i el treballador seràn el mateix peer, però seràn totalment independents.
4. El manager, per la seva part, cercarà el grup de monitors i com no el trobarà, el crearà i passarà a formar-ne part.
5. El peer enviarà la seva informació al manager i esperarà a rebre treballs.

- El segon cas és un cop ja hi ha una jerarquia creada, el següent usuari que es connecti al sistema i estigui a la mateixa àrea, seguirà la seqüència d'accions següent:

1. Cerca un grup de treballadors i el d'àrea, i com aquests ja estan creats, passen a ser membres.
2. Cerca el avís del manager, i un cop trobat passarà a enviar-li missatges amb la seva informació.

- Si un usuari que està en un altra àrea, es connecta al sistema amb la situació que ha quedat abans, farà el següent:

1. Cerca un grup de treballadors i com ja està creat, passarà a ser membre d'aquest.
2. Cerca l'àrea i com està en un àrea diferent de les altres arees ja creades, n'hi crearà un de nou.
3. Cerca l'avís del manager de l'àrea, i al igual que en el primer cas, al no trobar-lo, crearà el gestor.
4. El peer enviarà missatges amb la seva informació al gestor i espera a rebre treballs.

Suposant l'ingrés de dos peers a la mateixa àrea i un a una diferent, en aquest moment hi haurà tres treballadors al sistema, dos grups d'àrea amb un i dos peers respectivament, i dos gestors al grup de gestors.

Manteniment del sistema Durant el temps que s'està al sistema p2p, la resta de peers poden entrar i sortir. A més, sense aquest manteniment de la xarxa es poden donar situacions no desitjables, com per exemple dos gestors en un mateix grup d'àrea o informació de treballadors que no existeixen. Per tant, cal controlar l'estat del sistema periòdicament.

A més, cal que la informació dels treballadors estigui mínimament actualitzada, ja que es podria arribar a saturar un treballador. Per dur a terme aquesta tasca, cada peer s'aprofita dels missatges d'actualització d'informació per a mantenir la jerarquia del sistema.

Cada 20 segons els treballadors enviaran la seva informació al seu gestor i a la vegada comprovaran que aquest continua operatiu. Amb això es garanteix que sense sobrecarregar el sistema de missatges, els treballadors sempre depenguin d'un manager.

Per la seva part, els gestors fan comprovacions cada 20 segons de tot el sistema. Aquests han de mirar que els treballadors que ell controla estiguin operatius i també que no hi hagi altres gestors en el mateix grup. Per últim, han d'anar actualitzant la informació de la seva àrea a la resta de gestors de *CompP2P*, amb això també es comprova si els altres gestors estan operatius.

Seguint aquesta metodologia, el sistema està completament actualitzat i amb tota la funcionalitat en un marge de 40 segons sense estar saturat.

Sortida del sistema A l'hora de deixar el sistema, l'actualització de l'estat es produeix per events, com ja s'ha vist al manteniment del sistema. Els dos únics casos que hi ha, és si cau un treballador o si a més és gestor del grup d'àrea.

En el cas de que el peer que té el rol de manager surti, els peers que estan per sota d'aquest, no podràn enviar-li la seva informació i passaran a tornar a buscar un gestor. Es tornaria a la situació, ja explicada, de l'entrada a *CompP2P*.

I en el segon cas, si l'equip que deixa el sistema és un treballador, quan el seu antic gestor intenti enviar-li un missatge de comprovació i no pugui fer-ho, l'eliminarà de la seva memòria.

2.1.3 Gestor

Com ja s'ha presentat abans, la presència d'un gestor és necessària per optimitzar el sistema i minimitzar el número de comunicacions i d'informació emmagatzemada, però no és la seva única tasca. En la figura 2.1 està representat tant el gestor com el planificador que es detallen a continuació. A més, en aquest apartat s'explica les diferents tasques que ha de dur a terme un manager.

Un dels papers més importants és el del manteniment de la jerarquia. Cal que aquest rebi la informació que els treballadors li envien periòdicament i la emmagatzemi per tal d'utilitzar-la a l'hora de distribuir les tasques. Amb això, s'aconsegueix a la vegada que la informació es mantingui actualitzada que els altres peers sàpiguen si aquest segueix operatiu.

Per comprovar que els peers dels quals té informació continuen al sistema enviarà un missatge periòdicament igual que amb els missatges de comprovació d'estat de la resta d'àrees.

Per tal de comprovar si la jerarquia és correcta, també ha de controlar si hi ha altres managers a la seva àrea, en cas de trobar-n'hi, avaluarà si és més prioritari que l'altre, i en cas contrari deixarà de ser gestor. Tots els peers que depenien del gestor que ha deixat de ser-ho, passaran a dependre de l'altre.

A més del manteniment del sistema, la tasca més important que porta a terme un gestor és la distribució de les tasques que li arriben entre els diferents treballadors del sistema seguint diferent criteris d'assignació. Per això, el manager disposa d'un *planificador* (o *dispatcher*) on s'emmagatzema la informació de tots els treballadors de l'àrea i la informació global de les altres àrees.

A la figura 2.4 es representa gràficament un grup d'àrea amb els diferents rols dels peers que en formen part i la tasca del planificador.

2.1.3.1 Planificador

Aquest planificador guarda la informació de tots els treballadors controlats pel gestor i de les altres arees del sistema *CompP2P* per tal d'assignar les tasques seguint els seus criteris.

Cada cop que algún usuari vol executar una tasca al sistema, el dispatcher seleccionarà un treballador de la seva àrea, o si detecta una possible saturació de treball de la seva àrea, delegarà la feina cap a altres àrees.

En aquesta primera versió de *CompP2P*, el dispatcher aplica una assignació Round Robin, i el 10% (un de cada deu) el delegarà cap a altres àrees. El tipus d'assignació és fàcilment variable variant el mètode *dispatch()* del dispensador per facilitar la tasca a futurs desenvolupadors.

2.1.4 Comunicació

La part més important del sistema de peers és la seva comunicació. Ja sigui per l'enviament dels treballs a executar o pel manteniment del sistema. Aquesta capa de comunicació també està representada en la figura 2.1 situada al principi del capítol.

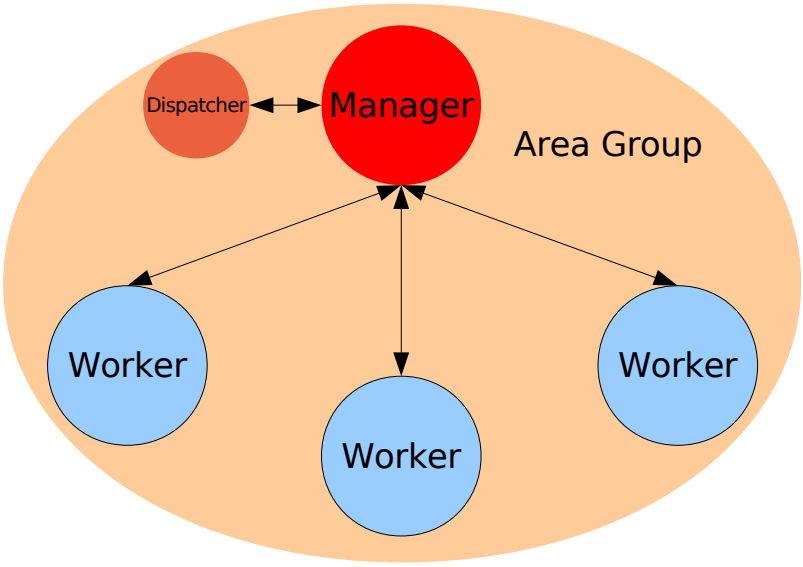

Figura 2.4: Grup d'àrea amb un gestor.

La forma de comunicació més eficient en JXTA és mitjançant pipes, enviant diferents tipus de missatges a través d'aquests. La manera més ineficient és declarar-les estàticament, però és més ineficient. Per això el sistema treballarà amb creació dinàmica dels pipes. Per a que els altres coneguin l'existència d'aquests, s'ha de publicar un avís en un mòdul.

Els gestors publicaran els seus pipes d'entrada, i aquests rebran la informació dels demés peers juntament amb els seus pipes d'entrada i l'emmagatzemaran per utilitzar en el futur.

2.1.4.1 Modes de comunicació

A JXTA l'obertura d'un pipe és un procés que pot ralentitzar les comunicacions, per aquest fet *CompP2P* introdueix una *caché de pipes* amb els que han estat oberts prèviament. Així si per exemple un peer que vol enviar 50 treballs a un altre peer, haurà d'obrir la comunicació una vegada i la resta solament haurà de bolcar el missatge. L'estat d'aquests peers serà comprovat periòdicament, i eliminats de la caché en cas de no ser operatius.

Aquest mecanisme té un desaventatge amb el tipus d'enviaments que utilitza JXTA, ja que aquest envia el missatge estigui o no disponible l'altre extrem del pipe. És per això que per comprovar l'estat del sistema no és útil. Per prevenir aquests casos s'introdueix l'enviament de missatges sense caché.

Un altre problema és la saturació de pipes, tot i haber eliminat la sobrecàrrega per enviar molts missatges a un mateix peer, es poden donar congestions i per això s'introdueix l'enviament amb reenviament en cas d'error.

Els tipus d'enviament de missatges són:

- Enviament sense caché ni reenviament.
- Enviament amb caché i sense reenviament.
- Enviament amb caché i reenviament en cas d'error.

Tot i les diferències, tots els mètodes intenten obrir el pipe utilitzant *retransmissió exponencial binària*. En cas de que l'obertura del pipe falli, s'esperarà un número aleatori entre 0 i 1 segon, a continuació entre 0 i 2 segons, seguidament entre 0 i 4 segons i així succesivament fins a un limit total que en cas de l'utilització de caché pot arribar a 31 segons i en cas contrari fins a 7 segons.

Pel que fa a l'enviament amb reenviament, en cas d'error s'afegeix retransmissió exponencial binària per fer aquest reenviament en aquest cas en dècimes de segon i fins a 31 decimes de segon.

Introduint aquests mecanismes s'aconsegueix opimitzar les comunicacions i eliminar la possibilitat de fallada en cas de saturació. A la vegada també es dona una sèrie de mecanismes diferents per tots els casos d'enviament de missatges: d'estat, d'informació, de treball i de gestió.

2.1.4.2 Missatges

Com ja s'ha dit, pels pipes de JXTA s'envien missatges de diferents tipus. A continuació, és descriuen els tipus de missatges que intercanvien cada tipus de peer i la informació que aquests contenen:

- **Peer a Manager**: Els missatges que envia qualsevol peer al seu Manager:

  PeerInfo aquest missatge conté la informació rellevant del peer. És enviada per part d'un peer cap al seu manager.

  ScheduleJob conté l'identificador del treball que és vol executar remotament junt amb el pipe d'entrada d'aquest peer.

- **Manager a Peer**: La informació que envia el manager a qualsevol peer de *CompP2P*:

  ManagerAlive enviat per part del manager a tots els peers de la seva area per informar de que està actiu.

  DispatchJob notifica a un peer de que ha sigut seleccionat com a treballador. Conté l'identificador del treball i el pipe del MasterPeer.

- **Peer a Peer**: Els tipus de missatges que intercanvien els peers de la xarxa, són:

  ChosenWorker quan un peer ha sigut seleccionat per un manager com a treballador, aquest envia l'identificador de la tasca i el seu pipe al MasterPipe.

  SendJob conté l'identificador de la tasca, els pipes del master i del worker, i la pròpia tasca a executar remotament. A més, conté la informació del treball per que el treballador pugui demanar el fitxer JAR en cas de no tenir-lo.

  FinishedJob igual que "SendJob" però amb la tasca ja realitzada.

  RequestJAR petició d'arxiu font d'un treball.

  SendJAR conté l'arxiu font d'una classe i a continuació el treball a executar.

- **Manager a Manager**: Els gestors dels diferents grups d'àrea intercanvien els següents tipus de missatges:

  ManagerInfo conté la informació especifica del gestor d'àrea, com per exemple la càrrega de treball o la quantitat de treballadors.

  ScheduleJob aquest missatge conté l'identificador del treball que és vol executar remotament junt amb el pipe d'entrada del peer master del treball en qüestió.

- **Globalment**: Entre tots els tipus de peers es poden intercanviar missatges per obtenir estadistiques del sistema:

  Management aquest missatge conté la comanda de petició d'informació i/o la resposta d'aquesta.

2.1.4.3 Dades

Els missatges que s'han descrit en l'apartat 2.1.4.2, contenen cert tipus de dades més complexes o més trivials segons el seu tipus. El cas dels treballs i de les estadístiques, per exemple, és més complex, són reutilitzats per peticions, respostes i transport d'informació segons el moment i el tipus de missatge que els continguin.
No sent així, per exemple, el cas de la informació dels peers o dels managers que sempre contenen les mateixes dades i en el mateix format.
A continuació passem a detallar els dos tipus de dades complexes i el funcionament global de cadascun:

1. **Job**: Sempre conté l'identificador de la tasca a la que fa referència.

   Aquest missatge pot contenir les dades del master del treball, les dades del treballador que executarà la tasca, el treball a executar, les fonts d'aquest treball, el seu nom, el path local, el hash de les fonts, el temps que ha estat en la cua de treballs , i el temps d'execució.

   La justificació de l'ús de tantes variables, és que amb un únic tipus de dades, que coneixen tots els peers és pot englobar un gran número de missatges sense haver de sobrecarregar la xarxa. Per exemple, si es vol sol·licitar un treballador, no cal que s'enviï tota la feina, sinó que amb la informació del master ja es suficient.

   Per emmagatzemar la tasca a la cua de feines, s'actualitzarà la informació complerta del treball, és a dir quan es posa a la cua el treball, únicament coneixem el master i l'especificació de la feina, però un cop es conegui el treballador designat, se li afegirà la seva informació.

2. **Management**: únicament és enviat en els missatges d'estadístiques. Tot i això aquest missatge pot ser de petició o de resposta. Per això consta d'un flag on s'especifica si es tracta d'una petició o d'una resposta, a més, conté la informació requerida per aquesta comanda.

   Amb això, disposem de la possibilitat de fer una possible multiplexació dels missatges de petició i resposta a versions futures, és a dir, una petició i una resposta en el mateix missatge.

2.1.4.4 Comunicacions entre peers

En aquest apartat es mostra els diferents tipus de comunicacions que hi ha a la xarxa del sistema de peers i en el moment en el que es donen.

1. **Manteniment de la jerarquia de grup**: Cada 20 segons cada peer treballador ha d'enviar un missatge amb la seva informació "PeerInfo" al manager de la seva àrea. Aquest emmagatzemarà aquesta informació i cada 20 segons enviarà a tots els peers de l'àrea un missatge de "ManagerAlive".

   L'enviament d'aquesta informació, evidentment no haurà de fer ús de la informació cachejada, per que el pipe podria figurar com a operatiu quan en realitat no ho és i enviar el missatge al buit. Per això, s'ha d'obrir el pipe per cada missatge de comprovació que es vulgui enviar.

2. **Manteniment de la jerarquia de gestors**: Per un altra banda, els managers també han de conèixer l'existència i la informació dels altres grups. Per això, cada 20 segons envia un missatge "ManagerInfo" a tots els gestors que integren el grup de monitors.

   Amb això, el que s'aconsegueix és una comunicació entre diferents arees. Podem veure un esquema dels dos exemples de comunicació en les figures 2.5 i 2.6.

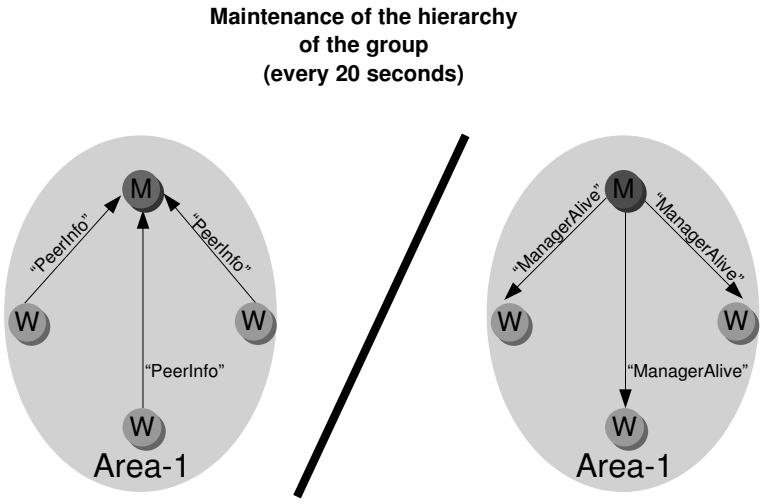

Figura 2.5: Missatges entre el mànager i treballadors d'un grup.

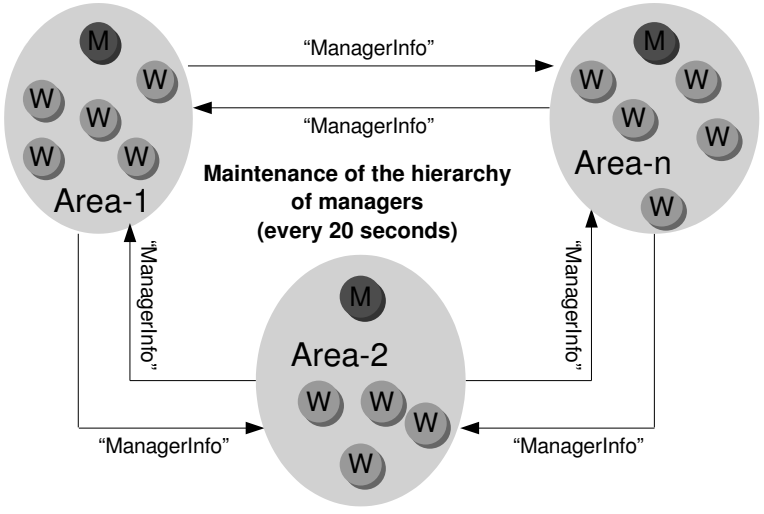

Figura 2.6: Missatges entre els mànagers dels diferents grups.

3. **Execució de treball remot**: Quan un peer vulgui executar un treball l'anomenarem *MasterPeer*.

En la figura 2.7 està representat gràficament aquest procés d'execució i a continuació s'explica detalladament com es realitza:

(a) Primer de tot, el *MasterPeer* enviarà un missatge "ScheduleJob" al gestor de la seva àrea per que aquest busqui un peer treballador (*WorkerPeer*).

(b) El manager seleccionarà el WorkerPeer i li enviarà un missatge "DispatchJob" per fer-li saber que ha sigut designat com a treballador. En cas de detecció d'una possible sobrecarrega aquest pot reenviar la sol·licitud d'execució remota d'un treball a d'altres àrees de treballadors i propagar el treball.

(c) El treballador quan rebi la notificació de que ha estat designat per ser el treballador per una certa tasca, avisarà del seu estatus al MasterPeer mitjançant un missatge de "ChosenWorker".

(d) Un cop el master ja té assignat el treballador per seva la tasca, li enviarà en un missatge del tipus "SendJob" amb el treball a executar.

(e) Quan el WorkerPeer rebi la feina procedirà a executar-la i un cop acabat li enviarà de tornada a MasterPeer amb un missatge "FinishedJob".

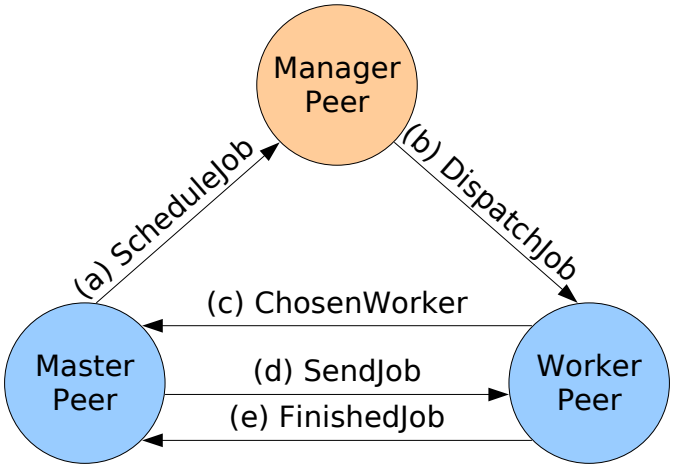

Figura 2.7: Missatges que intervenen en l'execució d'un treball.

Cal tenir en compte que a l'hora d'executar un treball, cal conèixer el treball a realitzar (elements pel que està format, funcions...). En cas de que a l'hora de dur a terme la tasca no disposi dels arxius fonts de la classe a executar, els hi demanarà al MasterPeer emprant el missatge "RequestJAR". Aquest li enviarà l'arxiu en un "SendJAR".

Si el gestor detecta que el seu grup de treballadors està sobrecarregat de feina, en comptes de seleccionar un treballador, propagarà el la petició de realització d'un treball a un altre grup, enviant un missatge "ScheduleJob" a un altre gestor.

2.1.5 Execució de treballs

En aquest apartat es descriuen tots els passos que ha de passar una tasca fins a ser executada i retornat el resultat a l'aplicació. A la figura 2.8 es mostra un esquema on es veuen tots els passos que fa un treball fins a ser executat.

1. Primer de tot l'aplicació envia la comanda "job "PATH" HASH" al dimoni TCP de *CompP2P*. Aquest interpreta que és un treball i mira si té les fonts per aquest arxiu descomprimides al seu directori de classes. Si no el té, el descomprimeix. A continuació espera a rebre un objecte d'una subclasse de Parallel serialitzat i quan el rep el passa al mòdul principal.

   Un cop al mòdul principal, es crea un objecte "Job" amb l'objecte a executar, identificadors i totes les dades importants per la seva execució. Aquest és l'objecte que es guardarà a la cua de treballs. A aquest peer se l'anomenarà màster de la tasca. Les característiques dels missatges ja s'han detallat a l'apartat anterior.

2. Una vegada el master té el treball per executar remotament, enviarà al seu gestor d'àrea un missatge amb la informació de la tasca.

3. Quan el gestor té la tasca a delegar, seleccionarà un peer (al que se li anomenarà worker) o continuarà delegant-la a altres gestors d'altres àrees.

4. El worker seleccionat notificarà llavors al master que és el peer seleccionat per executar aquella tasca.

5. Com ja s'ha detallat abans, el master enviarà aquest "Job" amb l'objecte de la classe ProxyParallel serialitzat per que sigui executat. Els arxius de fonts ja s'ha tractat anteriorment.

6. En aquest moment, el master espera a rebre el treball executat mitjançant un semàfor, per evitar així esperes actives. I un cop el worker torna la tasca executada al master, aquest el guarda a la cua de treballs. Aquesta cua en realitat sol és una cua als primers moments, és a dir actua com a cua FIFO per enviar els treballs al gestor, en canvi emmagatzema totes les dades dels treballs sempre.

7. Quan el treball ja està executat i a la JobQueue, es retorna al mòdul principal per que aquest a la seva vegada l'envii a l'aplicació mitjançant el dimoni TCP.

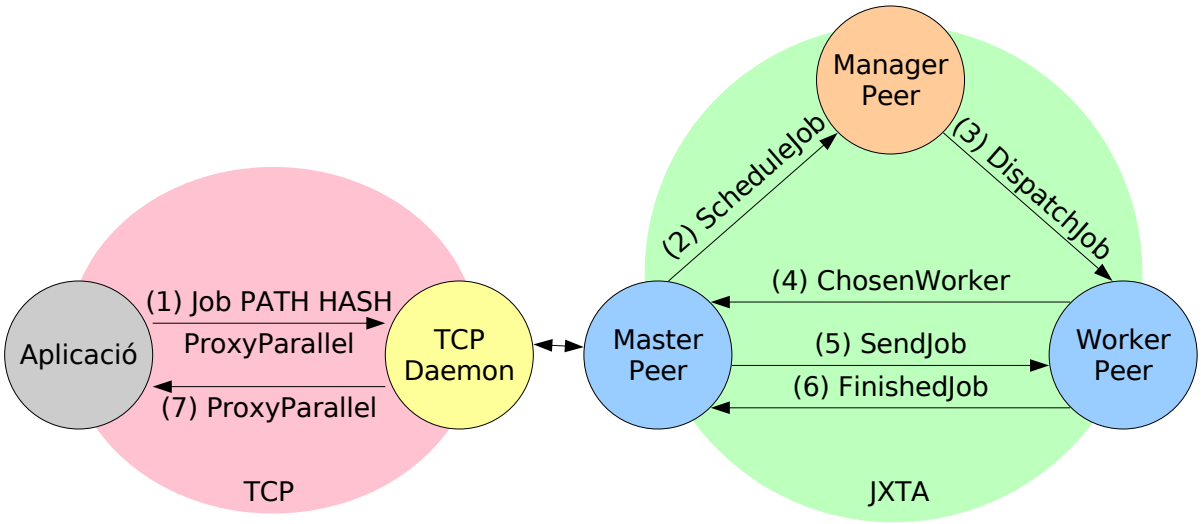

Figura 2.8: Execució d'un treball a CompP2P.

Capítol 3

CompP2P-GUI

CompP2P-GUI és una aplicació desenvolupada amb l'objectiu d'executar treballs remotament entre peers de forma gràfica i amigable per l'usuari. El seu desenvolupament ha estat dut a terme amb *Java-Swing*. S'executa com una aplicació amb interfície gràfica que permet portar a terme totes les funcions del programa[10] primitiu i afegeix noves funcionalitats.

3.1 Arquitectura

En aquesta secció s'explica el funcionament global de la interfície de còmput distribuït entre iguals, CompP2P-GUI. Tenint en compte els objectius marcats, es raonen i es presenten les solucions que implementa aquesta plataforma.

Era necessària una interfície amigable a l'usuari que no coneix a fons l'eina *CompP2P* o que, simplement, prefereix treballar amb una aplicació gràfica per tal de tenir més a mà tots els controls del sistema.

En els següents apartats, es donarà una visió detallada de cadascun dels aspectes fonamentals de l'arquitectura que utilitza CompP2P-GUI i la motivació d'aquestes solucions.

En la figura 3.1 es mostra un esquema on es pot apreciar l'arquitectura de l'aplicació. Al llarg del capítol s'explica detalladament cadascuna de les parts que composen els diferents nivells d'aquesta arquitectura.

3.1.1 Estàndards

CompP2P-GUI ha de tenir una funcionalitat "clàssica" d'una aplicació gràfica. És a dir, l'aplicatiu ha de permetre totes les opcions que contempla el dimoni *CompP2P*, és a dir: creació d'un nou peer o màster (dependrà de l'assignació que faci el sistema), executar tasques, visionar els resultats, veure les comandes del sistema ... etc. A més d'incorporar noves funcionalitats com és la que ens permet adaptar una aplicació qualsevol al paradigma *CompP2P*, la connexió remota ... etc.

Al tractar-se d'una aplicació gràfica, caldrà efectuar un manual que estigui a l'abast de l'usuari per tal de poder consultar les funcionalitats de cada botó i element de l'aplicatiu.

A més, l'aplicació ha de ser totalment multiplataforma, i per tant, cada plataforma, com a tal, té les seves particularitats. Una de les característiques identificatives més freqüents és el caràcter de separació d'arxius, així, per exemple als sistemes UNIX s'utilitza "/" i al sistema Windows "\". Per tant, a l'hora de crear els directoris, cal utilitzar la crida *System.getProperty*. Cal dir, però, que s'ha posat especial èmfasis als sistemes operatius que treballen amb *UNIX*, com són les distribucions de *Linux* i *Mac Os X*.

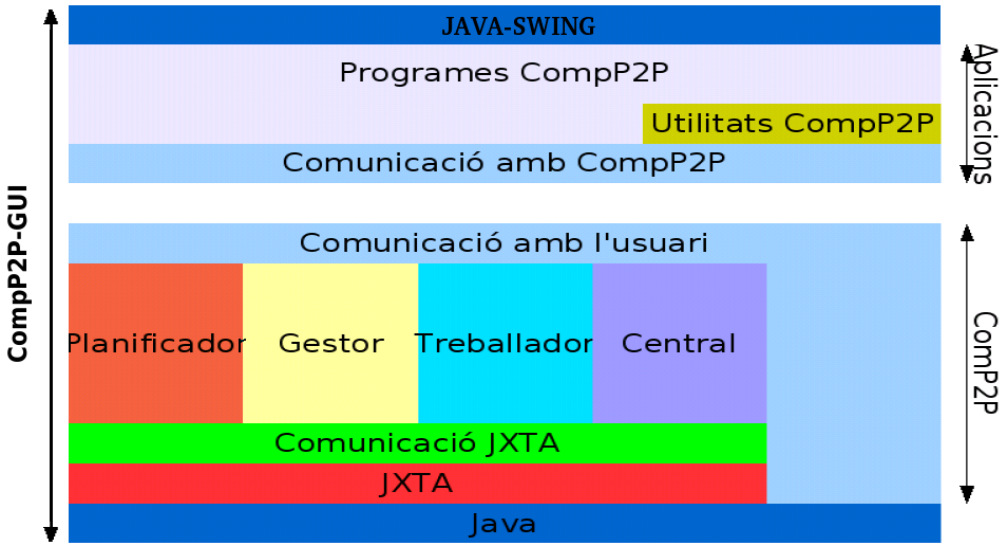

Figura 3.1: Esquema de l'arquitectura de CompP2P-GUI.

3.1.1.1 Threads

Un dels fonaments de *CompP2P* i *CompP2P-GUI* és l'utilització de fils d'execució, tal i com ja s'ha comentat en el capítol 2.

CompP2P i *CompP2P-GUI* ha de dur a terme més d'una tasca a la vegada, com pot ser rebre missatges de l'usuari, controlar l'estat del sistema peer to peer, executar treballs, mostrar l'ajuda de l'aplicació, executar altres programes dins de la mateixa màquina virtual, canviar l'aparença gràfica o *skin* de l'aplicatiu ...etc. És per això que l'ús de fils d'execució és un dels seus punts claus.

3.1.2 Disseny gràfic i funcional

Un requisit per una interfície gràfica, és que hi hagi totes les opcions necessàries, sense rutes confuses, disperses o intel·ligibles. CompP2P-GUI, ha intentat seguir un model que respecti els principis de l'enginyeria de software, i la facilitat pel que fa a l'usuari seguint també metodologies de creació d'interfície amb un ús estàndard.

3.1.2.1 Menú(s) principal(s)

Per dotar al sistema d'un aspecte simple i net, s'ha optat per la navegació entre temàtiques de l'aplicació per un element descendent de la classe *Object* de *Java*: *JTabbedPane*. Aquest és un objecte del tipus , llengüeta o "tab". Aquest tipus d'objecte gràfic es pot veure en la figura 3.2 en la que es veuen totes les llengüetes de l'aplicació.

Pel que fa als submenús de l'aplicació també s'han usat les llengüetes. S'ha respectat una llei en l'enginyeria de software en que precisa que tota aplicació gràfica no tingui una navegació per menús de més de 3 clics. Aquest aspecte es pot apreciar en la figura 3.3, en la que es mostra la navegació més llarga de l'aplicació: 3 clics.

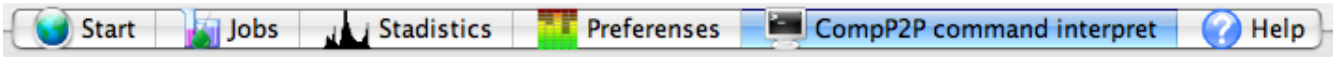

Figura 3.2: Tabs generals del sistema CompP2P-GUI.

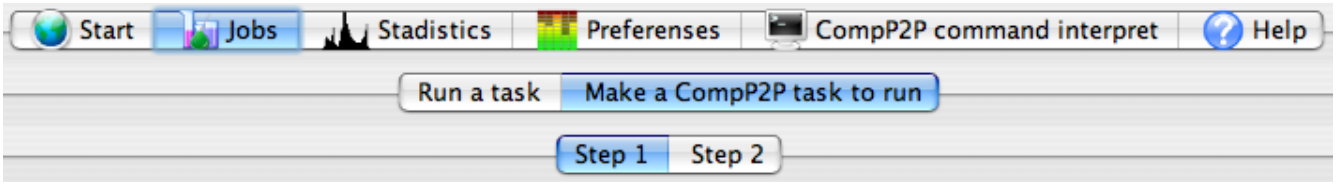

Figura 3.3: Navegació per tabs del sistema CompP2P-GUI.

3.1.2.2 Input(s) i Output(s) d'informació

Pel que fa a la introducció de la informació, així com la sortida d'aquesta de/a sistema *CompP2P*, s'usaran unes àrees de text (*JTextArea* o *JTextField*) que han intentat emular les terminals dels sistemes *Unix* i els camps dels formularis web, respectivament. En podem veure una mostra en la figura 3.4. Cal destacar també, la creació d'una taula en la que es mostren les estadístiques globals d'una execució d'una tasca; aquesta, es pot veure en la figura 3.5.

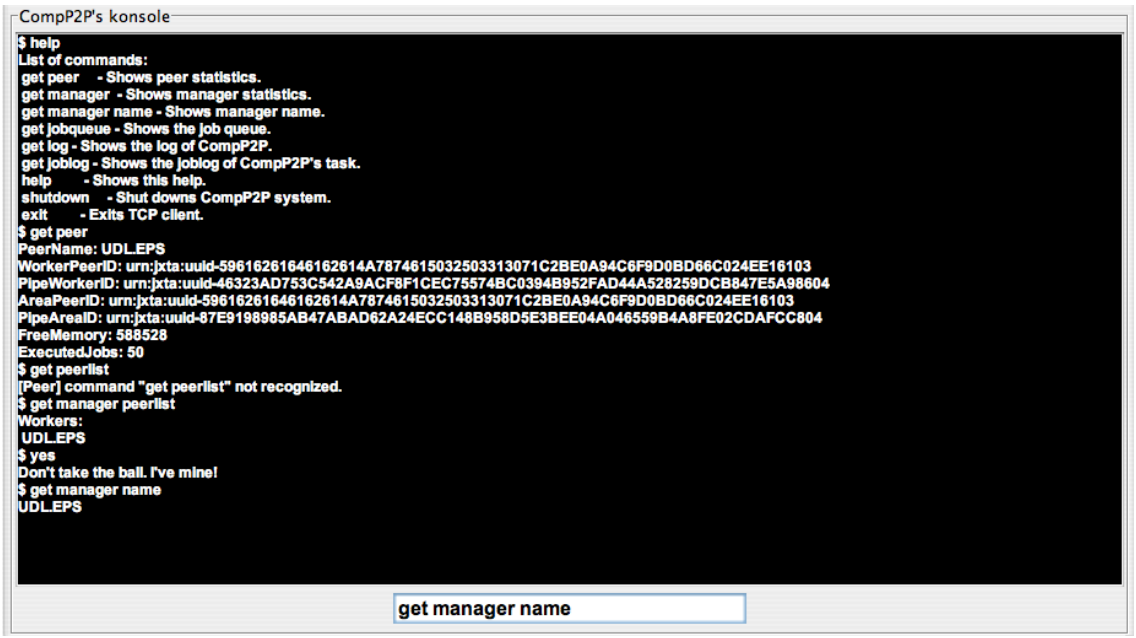

Figura 3.4: Sistema gràfic d'entrada/sortida del sistema CompP2P-GUI mitjançant l'ús d'àrees de text.

Table of statistics

JobID	WorkID	Master	Worker	ExecutionTi...	QueueTime
1162742...	3	UDL.EPS	UDL.EPS	0	242
1162742...	3	UDL.EPS	UDL.EPS	0	242
1162742...	3	UDL.EPS	UDL.EPS	0	210
1162742...	3	UDL.EPS	UDL.EPS	0	264
1162742...	3	UDL.EPS	UDL.EPS	0	251
1162742...	3	UDL.EPS	UDL.EPS	0	265
1162742...	3	UDL.EPS	UDL.EPS	0	165
1162742...	3	UDL.EPS	UDL.EPS	0	191
1162742...	3	UDL.EPS	UDL.EPS	0	167
1162742...	3	UDL.EPS	UDL.EPS	0	186
1162742...	3	UDL.EPS	UDL.EPS	0	194
1162742...	3	UDL.EPS	UDL.EPS	0	210
1162742...	3	UDL.EPS	UDL.EPS	0	190

Figura 3.5: Sistema gràfica d'entrada/sortida del sistema CompP2P-GUI mitjançant l'ús d'una taula.

3.1.2.3 Llegenda/s de les opcions

Per batejar els diferents panells de la interfície, és a dir, per no perdre's en l'aplicació, s'han usat petites llegendes en forma de text properes a l'objecte del qual es vol donar informació. Podem trobar dues vessants; per una banda trobem els *JLabel* (veure figura 3.6) que correspon als panells i àrees de text, i la funció *setToolTipText*, que s'usa quan es desitja que un botó mostri informació abans d'ésser pitjat (veure figura 3.7).

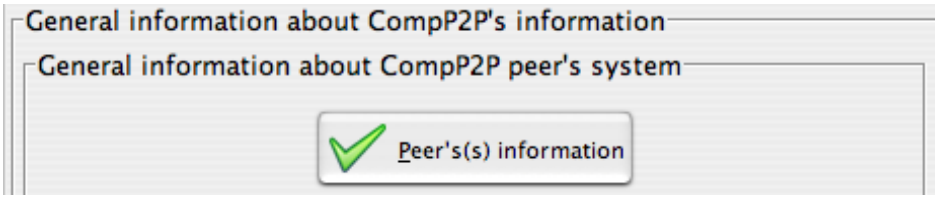

Figura 3.6: Informació del panell en que ens trobem.

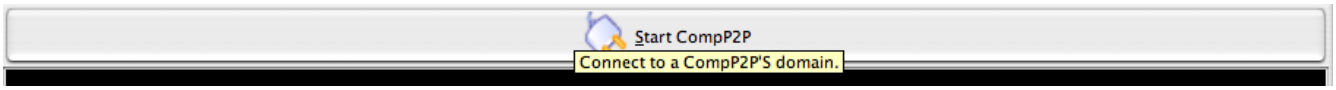

Figura 3.7: Informació referent al botó que es vol pitjar.

3.1.2.4 Accions principals

Per portar a terme una acció imperativa, és a dir, execució d'una tasca, modificació d'algun aspecte, mostrar l'ajuda ... en definitiva tot el que es tracta d'una acció concreta en l'aplicació d'importància rellevant (connectar, desconnectar, executar una tasca ... etc), s'ha decidit incloure en un botó (*JButton)* per tal de fer més intuïtiu l'ús de l'aplicació. En les figures anteriors com la 3.7 podem veure un exemple de botó.

3.1.2.5 Desplaçant la informació

Per desplaçar-se per la informació, ja sigui en les àrees de text o en la taula d'estadístiques, així com els mateixos panells de l'interfície, usarem elements desplaçables o *JScrollPane*, per tal que puguem gaudir d'un disseny més compacte i no perdem cap informació fent ús d'aquestes barres desplaçadores. En podem veure un exemple en la figura 3.8.

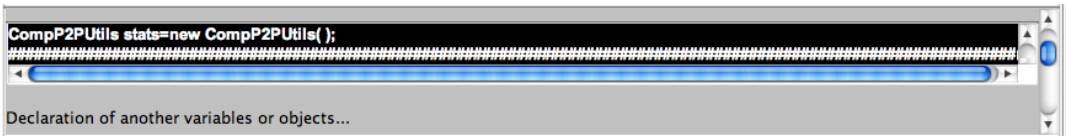

Figura 3.8: Barra lateral de desplaçament (horitzontal i vertical).

3.1.3 Estructura de l'aplicació

3.1.3.1 Descripció

Com ja hem comentat en l'esquema de la figura 3.1 dins l'apartat 3.1, l'aplicació gràfica ha d'ésser totalment funcional amb *CompP2P*, és a dir, tot el que permetia fer l'aplicació primitiva ha d'estar permès en la seva versió gràfica. A més a més s'han inclòs noves funcionalitats.

Així doncs, en els següents subapartats es comentaran quines classes composen l'aplicació, classificant-les per importància o tipologia.

Finalment es comenten els passos a seguir en cas de voler incloure quelcom més en la interfície gràfica.

3.1.3.2 Classes

En aquest apartat, s'exposaran totes les classes que composen l'aplicació comentat breument la finalitat de cadascuna d'elles. En la figura 3.9 podem veure que la interfície gràfica segueix una estructura jeràrquica, on hi ha classes que estan en capes superiors, d'altres que són "treballadores" i d'altres que simplement s'usen de connectors entre elles.

Així doncs, diferenciarem 5 tipus de classes:

1. Classe general: és aquella que conté a la resta de classes, podem dir que és la superclasse de l'aplicació. És la classe **CompP2PGUI**.

2. Classes temàtiques: són 6 classes, cada una d'elles cobreix una temàtica de l'aplicatiu. Dins de cada temàtica són la superclasse. Aquestes classes són: **Inici**, **Job**, **Statistics**, **Preferenses**[1], **Terminal** i **Help**.

3. Classes treballadores de primer ordre: són aquelles que s'han de crear per tal que l'interfície segueixi les funcions purament lligades a la idea primitiva de l'aplicatiu. Aquestes són: **Corre**, **Killer**, **LoadApp**, **Manager**, **Peer**, **Remot**, **Ajuda**, i **ShowManual**.

4. Classes treballadores de segon ordre: són aquelles classes que no es poden incloure dins les classes de primer ordre pel fet que no són de tanta importància. Aquestes són: **MakeFileApp**, **GeneralEnvironment**, **Parametres** i **ShowAuthors**.

[1]Tot i que sabem que preferències en anglés es tradueix com *Preferences*, no podem usar aquest nom de classe, degut a que *Java* ja el té reservat.

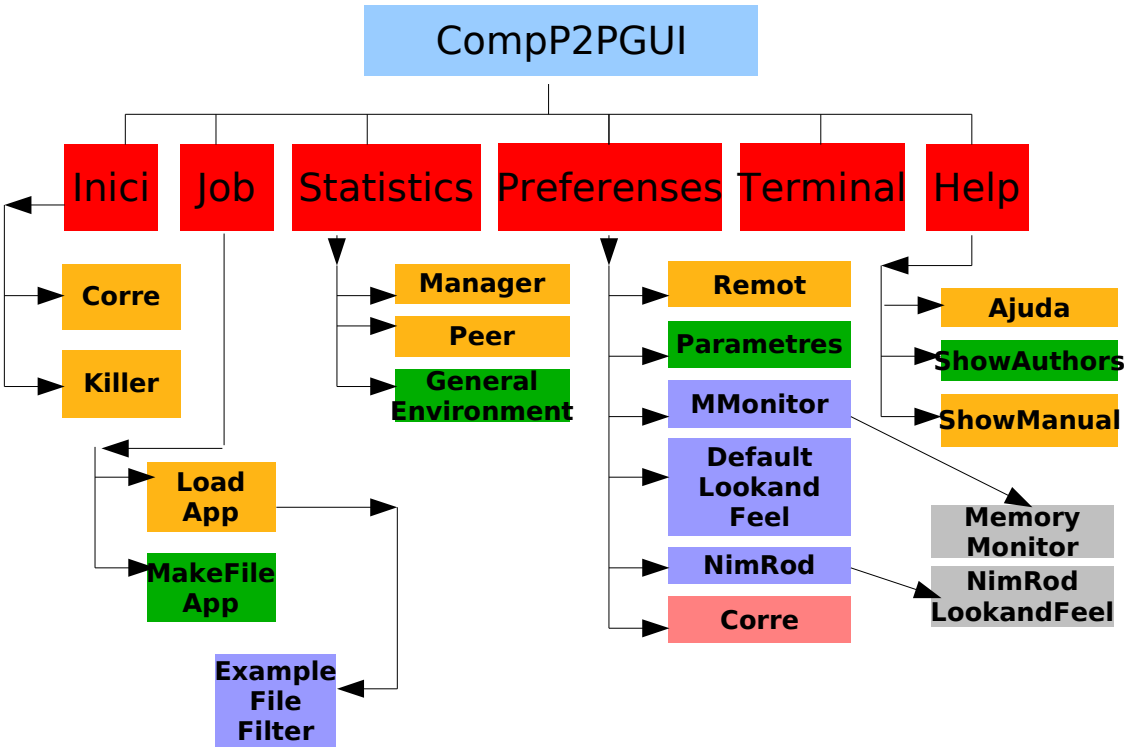

Figura 3.9: Estructura jeràrquica de classes de la interfície.

5. Classes auxiliars: són aquelles que s'han afegit a l'aplicació per dotar a l'aplicació de més opcions gràfiques i funcionals. Aquestes són: **ExampleFileFilter**, **MMonitor**, **DefaultLookandFeel** i **NimRod**.

Anem a veure cada classe quina funció porta a terme i els seus elements més rellevants:

Classe general

CompP2PGUI: és la classe principal que posa en comú i uneix tots els elements que composen l'interfície de l'aplicació. És a més, la classe encarregada de "llançar" l'interfície gràfica. En aquesta classe diferenciem 3 capes:

- Capa 1: aquella on es creen les 6 subclasses generals i s'afegeixen a la superclasse que és ella mateixa.
- Capa 2: es creen tots els mètodes que ens permeten obtenir tots els elements de la resta de classe que tenen certa importància pel correcte funcionament de l'aplicació.
- Capa 3: crea la finestra amb tots els components de l'aplicació amb dimensions relatives al tamany de la pantalla des d'on s'executa el programa.

Classes temàtiques

A continuació es farà un breu examen de les 6 classes que formen part d'aquest conjunt; diferenciem:

- **Inici**: és la classe que s'encarrega d'executar el *CompP2P*. També és l'encarregada de clausurar la comunicació. Finalment, ens permetrà veure la informació referent a l'inici, clausura i d'altres esdeveniments de l'aplicació mitjançant una àrea de text. Els tres elements fonamentals d'aquesta classe són:
  - Botó de connexió (Start CompP2P): s'usa per tal d'iniciar el procés d'autenticació i connexió a una xarxa **CompP2P**.
  - Botó de desconnexió (Disconnect CompP2P): serveix per desconnectar-se de la xarxa **CompP2P**.
  - Àrea de text en la qual es mostra informació relacionada amb els processos de connexió, desconnexió de **CompP2P** així com de l'execució de les tasques.
- **Job**: és la classe que s'encarrega de carregar *scripts* que tinguem emmagatzemats al sistema que permeten executar les tasques del tipus **CompP2P** així com l'assistència alhora de crear-ne de noves. A més a més mostra informació referent a l'execució d'aquestes tasques. Els elements fonamentals d'aquesta classe són:
  - La pestanyeta o llengüeta de carregar una tasca a executar (Run a task), conté les opcions que ens permeten carregar noves tasques a executar i visionar-ne els resultats; tindrem 3 elements fonamentals, són:
    - Botó per carregar un script que ens permetrà executar una aplicació; (Load application).
    - Àrea de text que mostra els resultats de l'execució de la tasca en format text; (Results).
    - Àrea de text que ens mostra la informació referent al *Dispatcher* de **CompP2P**; (Info Job).

    - Taula de resum d'estadístiques que mostra les estadístiques de l'execució portada a terme de forma dinàmica, és a dir només hi ha taula quan s'executa una tasca; (Table of statistics).
  - La pestanyeta o llengüeta per adaptar una aplicació normal a una compatible amb **CompP2P**; (Make a CompP2P task to run). Els elements fonamentals són:
    - Step 1: conjunt d'arees de text per crear el directori i els fitxers *java* de l'aplicació adaptada.
    - Step 2: àrea de text i botó que permet crear el script que llençarà la tasca creada amb anterioritat(abans l'usuari caldrà que compili les classes creades, veure capítol 5).

- **Statistics**: és la classe que ens permet veure informació referent al *manager* de la xarxa i també la nostra informació com a *peer*. A més a més podem conéixer informació referent a la màquina local, com pot ser versió de *Java*, directori d'execució ... etc. Els tres elements són:
  - Botó que ens mostrarà la informació del peer; (Peer's information).
  - Botó que ens mostrarà la informació del manager; (Manager's information).
  - Botó per mostrar la informació global del sistema; (Global environment variables).

- **Preferenses**: és la classe que ens permet iniciar una connexió del tipus **CompP2P** a un host de la xarxa local. A més a més permet variar el *Look&Feel* de l'aplicació, executar un monitor de memòria i determinar el nombre màxim de peers per tasca. Diferenciarem tres subapartats dins d'aquest panell; són:
  - Connexió remota a un host de la xarxa local mitjançant usant l'adreça *IP* del remot i el port *35557* (escollit per defecte pels autors del projecte "base").
  - Opcions del sistema *CompP2P*; trobarem el monitor de memòria i una opció que ens permet establir el nombre màxim de peers per tasca.
  - Opcions de l'interfície gràfica; podrem variar entre dos estils gràfics diferents: *NimRod* o el *Default* del sistema.

- **Terminal**: és la classe que ens permet usar les opcions *CompP2PUtils* del sistema. Per fer-ho disposarem de dos àrees de text. La inferior és un petit camp de text per introduïr ordres per tal que el sistema *CompP2P* les interpreti i ens mostri la sortida per l'àrea de text que hi ha al centre del panell

- **Help**: és la classe que usarem per trobar informació referent a comandes, autors i manuals de l'aplicació. Diferenciarem:
  - El botó d'ajuda (Show) que mostrarà les comandes disponibles de *CompP2PUtils* que poden ésser usades en la **Terminal** del *GUI*.
  - El botó dels autors (Authors) que mostrarà informació referent als autors de *CompP2P* i *CompP2P-GUI*.
  - El botó del manual (Manual) que ens mostrarà la documentació del projecte.

Classes treballadores de primer ordre

A continuació es farà un breu examen de les 8 classes que formen part d'aquest conjunt; diferenciem:

- **Corre**: és una classe que extén la classe *Thread* que s'usa per crear la rutina de connexió, és a dir, llença l'aplicació *CompP2P*. Conjuntament hi ha lligada amb aquesta classe la barra de progrés i el control de connexió, que verifica si aquella màquina (local) ja està connectada o no.
- **Killer**: és una classe que extén la classe *Thread* que s'usa per desconnectar-se de la xarxa *CompP2P*.
- **LoadApp**: és una classe que extén la classe *Thread* que s'usa per carregar un *script* que seleccionarà l'usuari d'un diàleg del sistema on podrà navegar pels directoris locals amb la finalitat d'executar la tasca (del tipus *CompP2P*) associada a aquell *script*.
- **Manager**: és una classe que extén la classe *Thread* que s'usa per mostrar la informació del *Manager* de la xarxa **CompP2P**.
- **Peer**: és una classe que extén la classe *Thread* que s'usa per mostrar informació referent al(s) *Peer*(s) del sistema.
- **Remot**: és una classe que extén la classe *Thread* que s'usa per connectar el *CompP2P* a una màquina remota de la xarxa local.
- **Ajuda**: és una classe que extén la classe *Thread* que s'usa per mostrar les comandes del sistema que estan a l'abast de l'usuari.
- **ShowManual**: és una classe que extén la classe *Thread* que s'usa per mostrar la documentació del projecte.

Classes treballadores de segon ordre

A continuació es farà un breu examen de les 4 classes que formen part d'aquest conjunt; diferenciem:

- **MakeFileApp**: és una classe que extén la classe *Thread* que s'usa per crear els fitxers i directoris necessaris per transformar una aplicació qualsevol a una compatible a *CompP2P*.
- **GeneralEnvironment**: és una classe que extén la classe *Thread* que s'usa per mostrar la informació global del sistema; aquesta informació fa referència a versions del compilador de la *Màquina Virtual de Java*, el sistema operatiu, el ditectori d'execució . . . etc.
- **Parametres**: és una classe que extén la classe *Thread* que s'usa per establir el nombre màxim de *peers* que executaran una tasca.
- **ShowAuthors**: és una classe que extén la classe *Thread* que s'usa per mostrar informació referent als autors de *CompP2P* i *CompP2P-GUI*.

Classes auxiliars

A continuació es farà un breu examen de les 4 classes que formen part d'aquest conjunt; diferenciem:

- **ExampleFileFilter**: és una classe que extén la classe *Thread* que s'usa per carregar el *script* que tenim dins de la nostra màquina per tal d'executar-lo.
- **MMonitor**: és una classe que extén la classe *Thread* que llença una altra aplicació *Java* que conté un monitor de memòria ubicat en la classe forana *MemoryMonitor*.

- **DefaultLookandFeel**: és una classe que extén la classe *Thread* que estableix el *Look&Feel* de l'aplicació al tipus *MetalLookAndFeel*.

- **NimRod**: és una classe que extén la classe *Thread* que estableix el *Look&Feel* de l'aplicació al tipus *NimRodLookAndFeel*, classe forana.

3.1.3.3 Afegir noves funcionalitats

En cas que es vulgui afegir noves funcionalitats, només caldrà modificar o afegir les classes en la seva posició jeràrquica, i enllaçar-la a la seva corresponent classe extesa *Thread* si es precisa portar a terme alguna operació que pugui disminuir el rendiment de l'aplicació gràfica.

Així doncs, anem a suposar que volem afegir una nova funcionalitat, aquesta tractarà d'afegir un monitor de la *CPU* de la nostra màquina. Anem a veure pas a pas el que hauríem de fer.

1. A quina temàtica pertany la nova incorporació?

   - La temàtica de la nova incorporació pertany a *Preferences*.

2. Quines relacions amb d'altres classes té la classe temàtica identificada?

   - Té relació directa amb 6 classes e indirectament amb dues més; així doncs, tindrem:
     - Relació directa amb les classes:
       * **Remot**
       * **Parametres**
       * **MMonitor**
       * **DefaultLookandFeel**
       * **NimRod**
       * **Corre**
     - Relació indirecta amb les classes:
       * **MemoryMonitor**
       * **NimRodLookandFeel**

3. Analitzar l'estructura de les funcionalitats d'aquella àrea temàtica i adaptar els coneixements per crear el nou contingut.

   - En aquest cas inclouríem un nou botó que posaria *CPUMonitor* proper al monitor de memòria i enllaçaríem l'acció del botó a una classe que extengués a la classe *Thread* per tal que llencés el monitor de la CPU.

Així doncs amb aquests exemple tindriem una estructura de classes semblant als que mostra la figura 3.10.

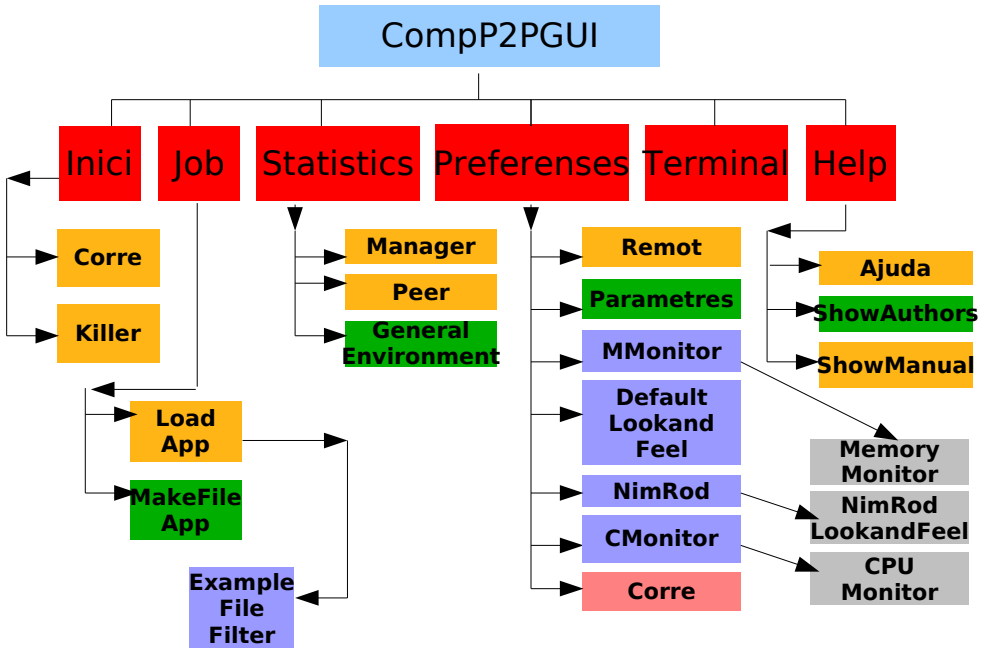

Figura 3.10: Estructura jeràrquica de classes de la interfície amb el monitor de CPU incorporat.

A continuació caldrà compilar la nova versió per tal que en la propera execució de *CompP2P-GUI* s'han afegit les noves funcionalitats, per fer-ho teclejarem en consola en la ruta mare de l'aplicació (./compp2p-gui/):

```
make gui
```

Aquesta comanda compilarà totes les classes relacionades amb l'interfície gràfica i ens deixarà aquesta en la darrera versió de l'aplicació, ja amb les noves funcionalitats incorporades. Per executar la nova versió, només serà necessari teclejar en la ruta que hem executat la comanda *make* (ruta *./compp2p-gui/*) el següent:

```
./rungui.sh
```

3.2 Funcionalitats adaptades

En aquesta secció es comentaran quines són les funcionalitats que s'han adaptat al sistema *CompP2P-GUI* sense haver dissenyat ni programat aquests mecanismes.

Una de les feines més importants d'un enginyer informàtic, és saber adaptar al treball propi, aquells mecanismes o funcionalitats que ja han estat programats per altres persones o organismes. Aquesta filosofia és una de les principals pilars de l'enginyeria en software lliure. Així doncs, anem a veure quines són aquestes funcionalitats que s'han adaptat i com ha estat aquests procés d'integració amb el projecte.

3.2.1 Aplicació *MemoryMonitor*

Memory Monitor[31], és un monitor de memòria del sistema creat per *Sun Microsystems*. Aquest monitor de memòria mostra gràficament i en temps real la següent informació:

- **Code Cache:** conté la memòria usada per la compilació i l'emmagatzematge del codi font.
- **Eden Space:** espai de memòria que s'assigna inicialment pels diferents objectes.
- **Survivor Space:** espai de memòria que després de la recol·lecció de brossa de la memòria encara és present en aquesta.
- **Tenured Gen:** espai de memòria que conté objectes permanents.
- **Perm Gen:** conté les dades de la Màquina Virtual de Java (JVM), incloses les classes i la memòria que aquesta usa.
- **Perm Gen [shared-ro]:** espai de memòria que conté dades de només lectura.
- **Perm Gen [shared-rw]:** espai de memòria que conté dades d'escriptura i lectura.

Aquest monitor de memòria dóna una idea general d'alguns paràmetres d'aquesta i també de la quantitat que la Màquina Virtual de Java està usant.

L'adaptació ha estat molt simple, primer hem ubicat l'arxiu *.jar* en el directori arrel de *CompP2P-GUI*, després hem associat a un botó de la interfíce gràfica una classe que hem implementat com a subclasse de la classe *Thread* per tal d'usar el llançador de Java i iniciar aquest monitor de memòria de forma concurrent a *CompP2P-GUI*. Tal i com és veu al codi següent:

```
class MMonitor extends Thread{
      public CompP2PGUI gui;
      public MMonitor(CompP2PGUI gui) {
             this.gui=gui;
      }

      public void run(){
                  try{
                      Process p = Runtime.getRuntime().exec("java -jar
                                                          ./MemoryMonitor/MemoryMonitor.jar");
                  }catch(Exception x){
                      JOptionPane.showMessageDialog(new JFrame(), x.getMessage(),
                                                "Error!",
                                                JOptionPane.ERROR_MESSAGE);
                  }
        }
}
```

3.2.2 Aplicació *NimRod*

NimRod[30] és una aplicació que conté un seguit de configuracions que usat adequadament ens permet personalitzar l'aparença gràfica de la nostra aplicació, és a dir, varia el *Look & Feel* de l'aplicació. És un

fitxer *.jar* que conté una estructura de classes java que caracteritzen un entorn gràfic especial i diferent al predeterminat. Aquesta classe ha estat extreta d'una pàgina web personal de *Nilo J. González*[2].

Adaptar aquesta aplicació ha consistit en situar el fitxer *.jar* en el directori arrel de *CompP2P-GUI*. Finalment, s'ha associat una casella de l'interfície gràfica on s'especifica si es vol usar aquest nou model gràfic. En cas que es seleccioni el model, és llençarà el codi d'una classe creada (extesa de la superclasse *Thread*) que executarà el codi que migrarà en temps d'execució l'aparença gràfica de l'aplicació. El codi d'aquesta classe auxiliar és:

```
class NimRod extends Thread{
      public CompP2PGUI gui;

      public NimRod(CompP2PGUI gui) {
             this.gui=gui;
      }

      public void run(){
                  try {
                       UIManager.setLookAndFeel("com.nilo.plaf.nimrod.NimRODLookAndFeel");
                       gui.repaintAll();
                  } catch (Exception x) {
                          JOptionPane.showMessageDialog(new JFrame(), x.getMessage(),
                                                        "Error loading Look and Feel!",
                                                        JOptionPane.ERROR_MESSAGE);
                    }
        }
}
```

3.3 Millores CompP2P

En aquesta secció, es comenten algunes de les millores que s'han introduït en el sistema original *CompP2P*. Aquestes millores, estan centrades bàsicament, en les comandes que accepta el sistema, és a dir, aquelles comandes que al gestor del sistema li permeten conéixer més profundament l'estat d'aquest. Anem a veure quines millores o ampliacions de les utilitats s'han efectuat.

1. Canvi del constructor per defecte de la classe *CompP2PUtils* per permetre les connexions remotes a peers remots dins d'una mateixa xarxa local. Per fer-ho, vam establir l'adreça de la classe per defecte al constructor, de la manera següent:

   ```
   public CompP2PUtils(){
          setAddress("localhost", 35557);
   }
   ```

   On *localhost* és l'adreça de l'equip remot passat com a paràmetre en la interfície gràfica i el port és el que usa el sistema per defecte (35557).

[2] Veure [30].

2. Canvi del constructor per paràmetres de la classe *CompP2PUtils* per permetre les connexions remotes a peers remots dins d'una mateixa xarxa local. Per fer-ho, vam establir l'adreça i port de la classe que rebem parametritzats de la interfície gràfica; ho vam aconseguir de la manera següent:

```
public CompP2PUtils(String host, int port){
       setAddress(host, port);
}
```

3. S'ha afegit el mètode *setAddress()* en la classe *CompP2PUtils* que és usat en els constructors d'aquesta classe. El codi d'aquesta funció és:

```
public void setAddress(){
       setAddress("localhost", 35557);
}
```

   On *localhost* es un paràmetre passat per l'usuari que usa la interfície gràfica, i el port és el que usa el sistema per defecte (35557).

4. S'ha creat un altre mètode *setAddress(String, int)*, aquest cop amb paràmetres que el que fa és enllaçar l'adreça i el port que l'usuari de la interfície gràfica introdueix amb l'adreça i el port de la connexió remota a un peer d'àrea local.

5. S'ha modificat la classe *CompP2P* per tal que accepti noves comandes de control del sistema, aquestes variacions, concretament han estat:

   (a) Hem afegit un mètode que captura els logs del sistema, tan els de connexió/desconnexió, execució de tasques o distribuïdors d'aquestes (*Dispatcher*). La funció en qüestió és:

```
public void setLogs(){
     try {
jobLog=new PrintStream(new FileOutputStream("joblog"));
log=new PrintStream(new FileOutputStream("log"));
System.setOut(log);
     } catch(Exception e){ }
}
```

   On *joblog*, és una nova comanda introduïda que permet veure l'execució d'una tasca. Aquesta comanda ha estat molt útil alhora de mostrar la informació en la interfície gràfica.

   (b) S'ha creat un nou mètode en la classe *CompP2P* anomenat *readJobLog* que és l'encarregat de llegir i retornar l'estat de l'execució d'una tasca. Tal i com hem comentat aquestes funcions han estat vitals per un desenvolupament integrat de *CompP2P* i *CompP2P-GUI*. El codi de la funció és:

```
private String readJobLog(){

String in, fin="";
try {
File f = new File("joblog");
```

```
FileReader aux = new FileReader(f);
BufferedReader br = new BufferedReader(aux);
if (f.canRead()){
    in = br.readLine();
    while (in!=null){
    fin += in+"\n";
    in = br.readLine();
    }
} else System.out.println("Impossible llegir el fitxer");
     } catch (Exception x){ System.out.println("Error al obtenir el joblog"); }
return fin;
}
```

Aquestes, doncs, són les millores més significatives que s'han dut a terme. El fet de modificar codi de l'aplicació primitiva, ha permés a la vegada, conéixer millor com funciona aquestal.

3.4 Comunicació entre CompP2P i CompP2P-GUI

La dificultat del projecte, incrementa, quan no solament s'ha de dissenyar i programar una interfície gràfica, sinó que a més a més, aquesta, s'ha de comunicar amb una capa base ja programada amb la seva pròpia interficie, la qual cosa implica haver de conéixer tot el sistema per poder dur a terme ampliacions. Per tant calia fer una tasca de documentació prèvia.

Primerament, cal disenyar la fusió i connexió entre els dos sistemes, és a dir, plantejar tot el procés des de que es pitja el botó que llança una tasca fins que aquesta s'executa al sistema base *CompP2P*. A més a més, caldrà recollir-ne els resultats per finalment tractat-los i mostrar-los a *CompP2P-GUI*.

Tot seguit mostrarem com és la connexió entre els dos sistemes de forma general. En la figura 3.11 es mostra què succeeix quan es pitja sobre un botó que té una acció que precisa una comunicació amb *CompP2P* (en aquest cas s'indica el flux de *CompP2P-GUI* cap a *CompP2P*). Anem a desglossar cada pas per tal que l'esquema sigui més entenedor.

1. Es pitja sobre un botó.

2. Aquest botó té associat un *action_Command* que s'activarà quan el botó s'hagi pitjat.

3. Aquest *action_Command* crearà una instància de la classe que ha de portar a terme l'acció que a la vegada és una classe que extén a la classe *Thread*,és a dir, que és una subclasse d'aquesta.

4. Ja dins del nou codi de la classe, es poden portar a terme dos tipus d'enllaços, aquests són:

   (a) Directe. És el cas que es dóna quan s'usa la classe *CompP2PUtils*, ja que simplement es crea un objecte d'aquesta classe.

   (b) Indirecte. El cas de la resta de classe que usem (*CompP2P, MMonitor, NimRod, Task*). És crea un procés en Java (*Process*), que llença la classe associada i rep el resultat de l'execució.

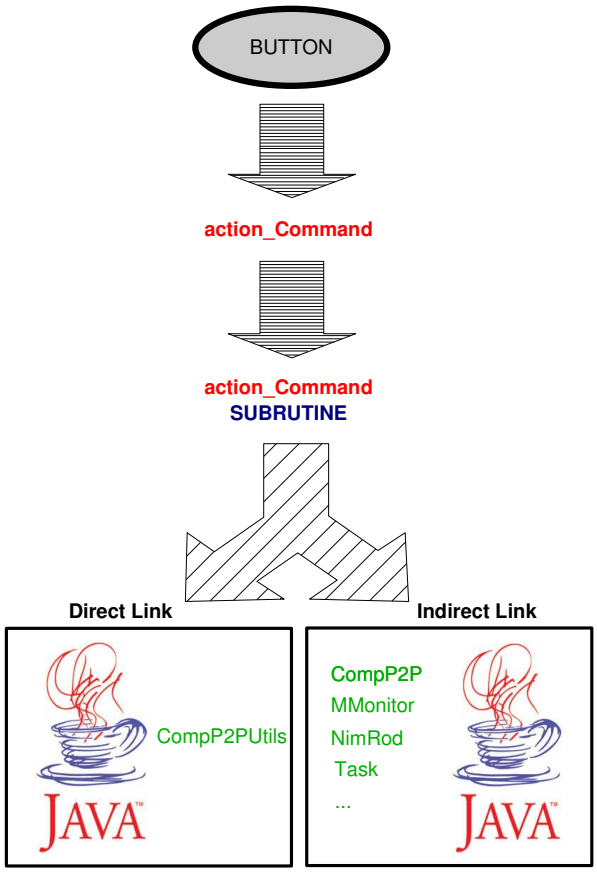

Figura 3.11: Esquema del flux d'esdeveniments que es dóna quan es pitja un botó.

En la figura 3.12 es mostra què succeeix quan es pitja sobre un botó que té una acció que precisa una comunicació amb *CompP2P* (en aquest cas s'indica el flux de *CompP2P* cap a *CompP2P-GUI*). Anem a desglossar cada pas per tal que l'esquema sigui més entenedor.

1. En el cas que tinguem un enllaç directe entre *CompP2P* i *CompP2P-GUI*, és a dir que la informació provingui d'una crida de la classe *CompP2PUtils*, tindrem:

   (a) Es parsejar la informació de *CompP2PUtils* directament a un *String*, *String*, arrai de *String/Integers*.

   (b) Aquesta informació es mostra directament a l'interfície gràfica.

2. En el cas que tinguem un enllaç indirecte entre *CompP2P* i *CompP2P-GUI*. La tasca haurà estat llançada usant un procés Java (*Process*). Aquesta tasca desarà la informació associada al procés. Segons el procés efectuarem:

   - *CompP2P*; el procés anirà rebent la sortida del sistema que s'emmagatzema en el fitxer *log* del mateix. Si tot és correcte, aquesta informació serà mostrada en la interfície gràfica. Si hi ha un error, és mostrarà un missatge d'error en la interfície gràfica.

- *NimRod* i *MMonitor*; el procés llença el fitxer java associat a aquestes classes i no fa res més, el codi d'aquestes classes s'executarà i ja no hi haurà més interacció amb la interfície gràfica.
- *Task* o les tasques que s'executen en el sistema, rebran el mateix tracte que en el cas de *CompP2P* a diferència que el el que el sistema va retornant diferents tipus d'informació que s'haurà de tractar de forma més elaborada degut a les maneres de mostrar la informació (taules, àrees de text ... etc.).

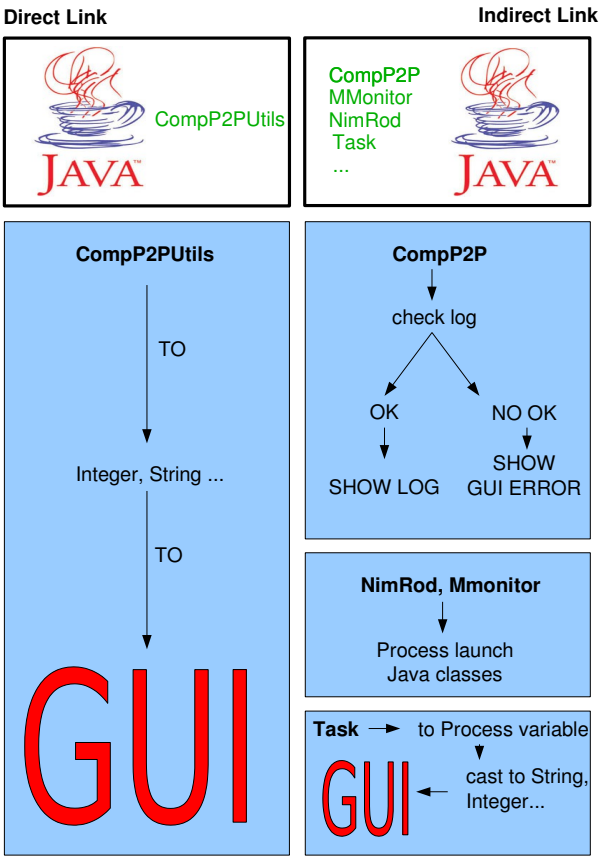

Figura 3.12: Esquema d'esdeveniments per rebre la informació associada al pitjar un botó.

Capítol 4

Manual CompP2P-GUI

Un cop definida la base i l'estructura de la interfície gràfica, tot seguit explicarem el funcionament de l'aplicació a mode de manual d'usuari.

4.1 Requisits del sistema

Primerament, necessitarem un ordenador connectat en una xarxa *LAN* amb connexió a internet. A més a més haurem d'usar un sistema operatiu que soporta la *JVM* o màquina virtual de java, concretament la versió 1.5 o posteriors. Així doncs, anem a veure com instal·lar en un entorn *Linux* aquest software.

4.1.1 Instal·lant Java 1.5

En aquest apartat s'explica com instal·lar l'entorn d'execució de Java per qualsevol sistema *Linux* per tots els usuaris del sistema, també és possible instal·lar l'entorn localment però es tractarà des d'un punt de vista global del sistema.

Per una instal·lació global caldrà tenir permisos de superusuari, per tant:

```
$ su
Password: *****
```

Cal seleccionar un directori per la instal·lació, per exemple */usr/java* i donar permissos d'execució a l'auto-extraible de Java:

```
# mkdir /usr/java
# mv j2re-1_5.bin /usr/java
# cd /usr/java
# chmod a+x j2re-1_5.bin
```

Ara cal començar el procés d'instal·lació:

```
# ./j2re-1_5.bin
```

Es mostrarà la llicència i cal acceptar-la. La instal·lació haurà començat i quan aquesta acabi al directori */usr/java/j2re1.5* o similar contindrà la instal·lació. En aquest moment ja es disposa de l'entorn Java 1.5.

Per l'execució de *CompP2P-GUI* sol és necessària la instal·lació de la màquina virtual i no cal configurar cap altre programa. El procés d'instal·lació complet es troba a [2].

4.2 Desempaquetant i executant CompP2P-GUI

CompP2P-GUI no requereix ser instalat per funcionar ja que és interpretat directament per la màquina virtual de Java. Per tant, n'hi ha prou descomprimint un fitxer que conté tot l'aplicatiu:

```
# tar xvfz CompP2P-GUI.tgz
```

Això crearà la carpeta *CompP2P-GUI* en el directori que s'ha executat la comanda i extraurà tots els fitxers dins.

Finalment, per executar cal fer:

```
# cd CompP2P-GUI/compp2p-gui/
# ./rungui.sh
```

A partir d'aquí, ja hem iniciat l'execució de *CompP2P-GUI* des d'on podrem gestionar el sistema *CompP2P*, tal i com es veu en la figura 4.1.

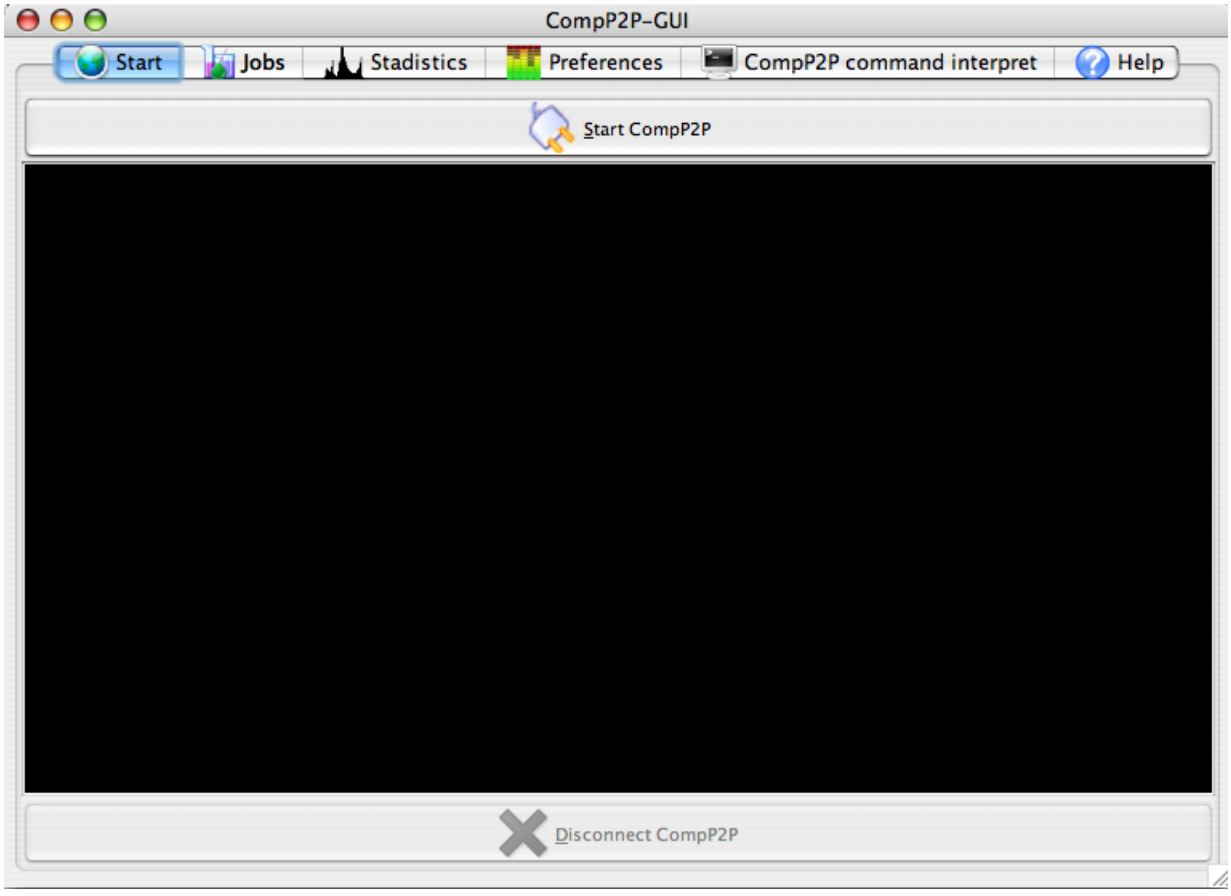

Figura 4.1: CompP2P-GUI funcionant.

4.3 Tab Start

Per iniciar la connexió amb el sistema cal que pitjem sobre el botó **Start CompP2P**, a continuació apareixerà una finestra per autenticar-se tal com la de la Figura 4.2. Només cal posar el nom amb el que

es vol identificar el nostre *peer* dins el sistema *CompP2P*. El *password* apareix per defecte (123456789), però es pot introduir qualsevol altre password.

Figura 4.2: Loguin a CompP2P.

Un cop autenticats, la barra de progrés de connexió anirà mostrant l'estat actual mentre per la consola de fons negre del mateix panell mostra el *log* de connexió.

Per desconnectar-nos del sistema *CompP2P*, cladrà pitjar el botó **Disconnect CompP2P**. Al cap duns segon de pitjar el botó ens mostrarà un missatge en el qual s'informa que el sistema s'ha desconnectat de forma exitosa.

Per tancar l'aplicació caldrà pitjar la creueta del marge superior dret de la mateixa aplicació.

4.4 Tab Job

És la pestanya que conté funcionalitats referents a l'execució de tasques així com de l'adaptació d'aquestes. Així doncs dins d'aquesta pestanya diferenciarem dues opcions més: carregar una tasca o adaptar-ne una. Anem a veure com funciona cada una d'elles per separat.

4.4.1 Executar una tasca

Per executar una tasca és necessari que es cumpleixin certs requisits, aquests són:

1. Tenir l'aplicació adaptada a *CompP2P* seguint l'exemple que trobem en el capítol 5.
2. Ubicar els arxius *Aplicacio.class* i *runap.class* a la carpeta mare de l'aplicació *CompP2P-GUI*, és a dir la carpeta *./compp2p-gui/*.

Un cop cumplim aquests requisits, anirem a la subpestanya **Run a task**. A continuació s'obrirà un quadre de diàleg en el qual es mostra les carpetes del nostre sistema partint de la carpeta de l'usuari. Es tractarà de dirigir-se a la carpeta arrel de **CompP2P-GUI** i carregar el *script runap.sh* que hi hem ubicat adequadament. A continuació pitjarem sobre el botó *Open* del quadre de diàleg.

Aquest procediment executarà l'aplicació que va lligada al *script*. La informació de retorn d'aquesta execució, la podem veure desglossada en:

1. Àrea de text batejada amb el nom **Result** en el qual trobem el resultat parcial (o total depenent de l'aplicació) de cada càlcul que s'efectua en l'aplicació.
2. Àrea de text batejada amb el nom **Info job** en el qual hi trobem informació de cada execució parcial referent a:
   - nombre de tasca
   - nombre del treball
   - manager del grup
   - peer que executa
   - temps emprat en l'execució
   - temps que ha estat en la cua
3. Taula amb el nom **Table of statistics** en la qual es mostra la informació de l'anterior àrea de text en format taula.

4.4.2 Adaptar una tasca

Seguint el patró d'adaptació que podem veure en el capítol 5, hem creat un guió per tal que l'usuari pugui adaptar una aplicació a *CompP2P* en la mateixa interfície gràfica; si més no aconsellement amb ímpetu la lectura d'aquest capítol per entendre millor el funcionament d'una aplicació compatible amb *CompP2P*.

Primerament caldrà seleccionar la pestanyeta amb nom **Make a CompP2P task to run**. A continuació farem el primer pas de l'adaptació, per això seleccionarem la pestanya **Step1** que trobem just davall de l'anterior llengüeta. Tot seguit caldrà substituïr, modificar o afegir en cada àrea de text el codi pertinent a l'aplicació així com editar totes les cadenes de caràcters on hi posa "*Class_Name*" pel nom de la classe que volem usar.

Un cop modificat pertinentment el codi de les diferents àrees de text, ens situarem sobre la petita àrea de text en blanc que hi ha a la part inferior. Dins d'aquesta àrea de text teclejarem el nom de l'aplicació (el mateix que hem usat per substituïr la cadena de caràcters "*Class_Name*". Tot seguit podem crear la carpeta amb aquest nom introduït si pitjem el botó **Generate the class directory**. A continuació podem ubicar els arxius necessaris dins d'aquesta nova carpeta pitjant el botó que té just a continuació

amb nom **Generate the application's files**. Havent seguit els passos, tenim una carpeta dins del directori arrel de *CompP2P-GUI* que conté tots els fitxers necessaris per l'aplicació que hem creat.

Tot seguit caldrà compilar aquesta nova aplicació creada[1]. Anem a veure un exemple de creació d'una aplicació amb nom *BenderCalc*:

1. Ens situem dins del la pestanya **Jobs** a continuació dins de la pestanya **Make a CompP2P task to run**, finalment en **Step 1**.
2. Editem adequadament el codi que es presenta i que correspongui al codi de la nostra aplicació general.
3. En l'àrea de text inferior escriurem: **BenderCalc**.
4. Pitjem sobre el botó **Generate the class directory**.
5. Pitjem sobre el botó **Generate the application's files**.

Finalment, caldrà crear un fitxer *script* en el qual es llança l'aplicació creada i compilada per l'usuari tal i com es mostra en el capítol 5. Els scripts *.sh* seran el tipus de "llençadora" que reconeixerà el filtre de carregador d'arxius del programa.

Per crear aquest fitxer caldrà que seleccionem la pestanya amb nom **Step 2**. Un cop dins del nou panell, trobarem una àrea de text de fons negre, en la qual ja hi tenim codi escrit; només caldrà modificar aquestes línies per tal que es llanci correctament l'aplicació que hem creat en el pas anterior. Per això caldrà substituïr nòvament els strings on hi posa "*Class_Name*" pel nom de l'aplicació (*BenderCalc* seguint l'exemple anterior). Finalment pitjarem sobre el botó **Generate** per tal de crear aquest script amb nom *runap.sh* (o un altre nom) en el directori abans creat.

Per tal d'executar aquesta nova aplicació, caldrà que movem els arxius *runap.sh* i *Aplicacio.class* al directori arrel de *CompP2P-GUI* i seguir després els passos indicats per executar una aplicació vist en 4.4.1.

Aquestes opcions que hem comentat es poden veure en la figura 4.3 i 4.4.

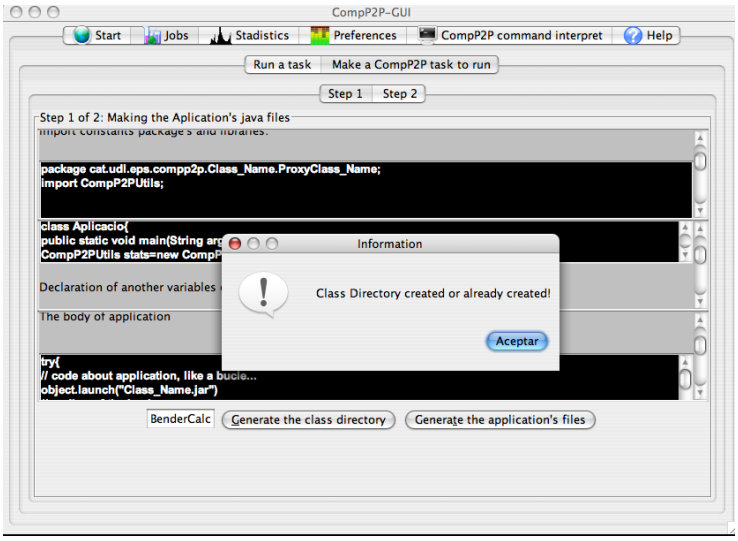

Figura 4.3: CompP2P-GUI pestanya *Tab Job*; creació dels fitxers .java i el directori de classe.

[1] Veure capítol 5.

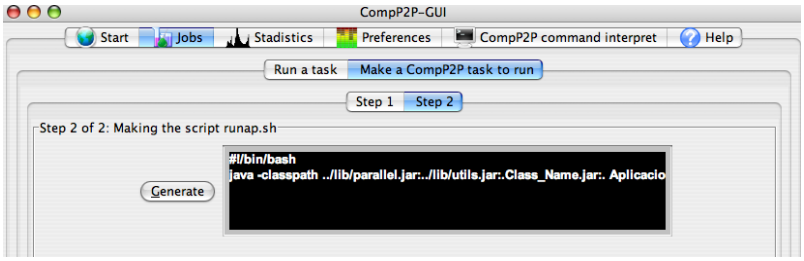

Figura 4.4: CompP2P-GUI pestanya *Tab Job*; creació del script "lençadora".

4.5 Tab Statistics

En aquesta pestanya tenim 3 botons, a continuació anomenem quin nom reben i quin tipus d'informació aporten:

1. **Peer's information**: conté informació referent al peer local del sistema, concretament:
   - Nom del peer local.
   - *WorkerPeerID*, *PipeWorkerID*, *AreaPeerID*
   - Memòria lliure.
   - Tasques executades.
2. **Manager's information**: conté informació referent al manager del sistema, concretament:
   - La mateixa informació del peer (un mànager no deixa d' ésser un peer).
   - Nombre de peers que composen el grup.
   - Nombre de mànagers del grup.
   - Memòria lliure del grup.
   - Nombre de tasques executades.
   - *DispatchedJobs*.
3. **Global environment variables**: conté informació del sistema, podem trobar-hi:
   - Directori d'instal·lació de *Java*.
   - Versió de la *Màquina Virtual de Java*.
   - Altra informació referent a *Java* i a la seva màquina virtual del nostre sistema.
   - Nom del sistema operatiu, arquitectura d'aquest i versió.
   - Nom de l'usuari, el directori *home* d'aquest, i el directori actual d'execució.

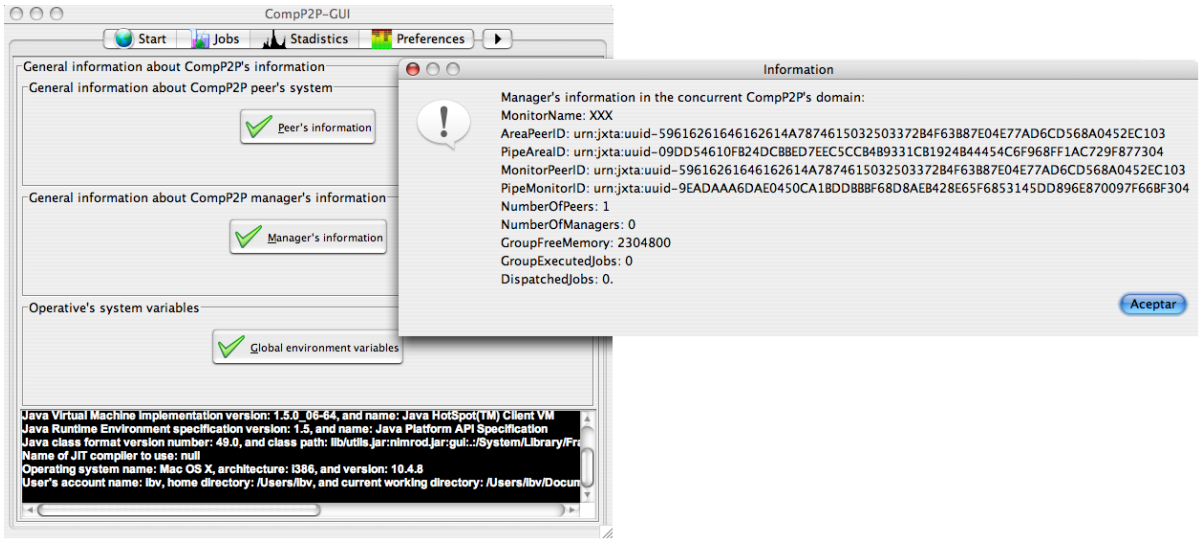

Figura 4.5: CompP2P-GUI pestanya *Statistics*.

Podem veure una captura en la figura 4.5 en que es mostra la informació del mànager i les variables generals del sistema.

4.6 Tab Preferences

En aquesta pestanya tenim 3 conceptes a tractar, anem a veure quines preferències o opcions podem modificar:

1. Opció de connexió remota a un host de la xarxa *LAN* local. Per connectar-nos, haurem d'introduïr l'adreça *IP* del host remot i el port (per defecte el port serà el 35557) i pitja el botó de connectar tal i com es mostra en la figura 4.6.

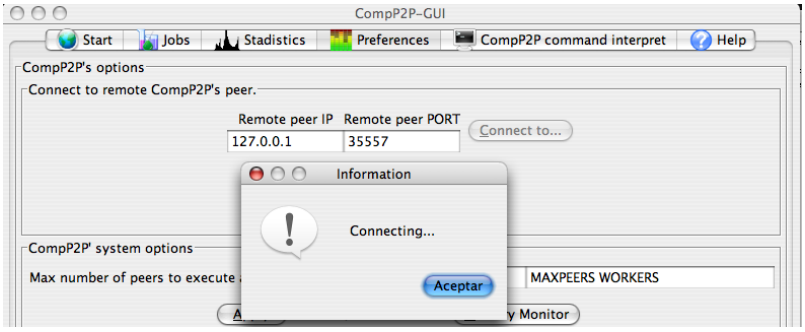

Figura 4.6: CompP2P-GUI pestanya *Preferences*; connexió remota a un host local.

2.a Opció que ens permet establir el nombre màxim de peers per treball. En la petita àrea de text on hi posa *MAXPEERS JOB* hi posarem el nombre màxim de tasques per un peer (en número), i en la següent àrea de text on hi posa *MAXPEERS WORKERS* hi posarem el nombre màxim de peers per una tasca (en número); a continuació pitjarem el botó **Apply**. Ho podem veure en la figura 4.7.

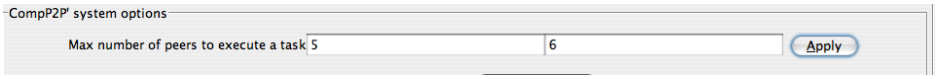

Figura 4.7: CompP2P-GUI pestanya *Preferences*; establir nombre màxim de peers per una tasca.

2.b Opció que llença un monitor de memòria. Cal pitjar el botó amb nom **Memory Monitor**. Un cop tanquen el monitor de memòria, no podrem tornar a llençar-lo fins una nova execució de *CompP2P*. Podem veure una mostra del botó i del monitor en la figura 4.8.

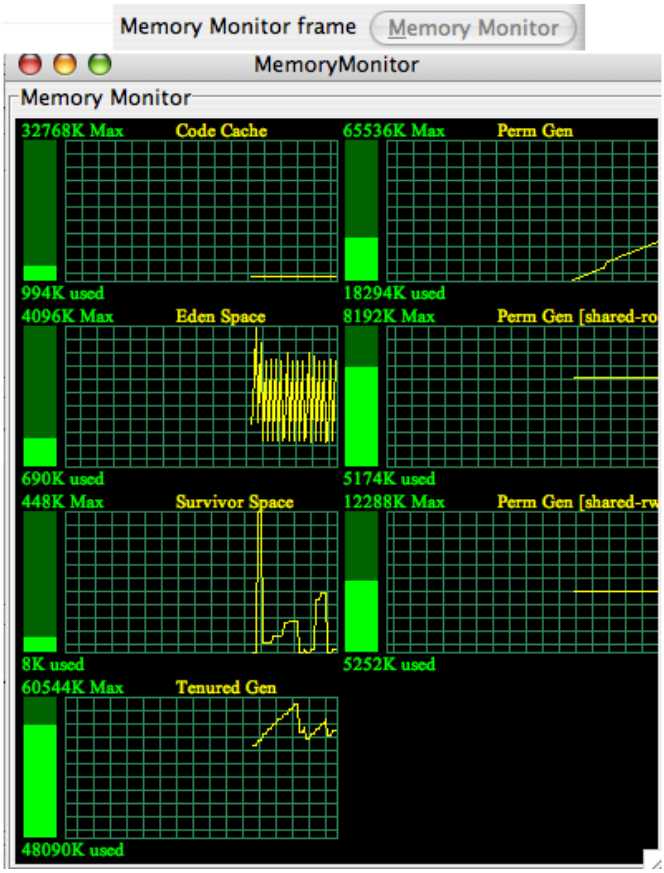

Figura 4.8: CompP2P-GUI pestanya *Preferences*; monitor de memòria.

2. La darrera opció ens permet canviar el *Look&Feel* de l'aplicació, és a dir, el disseny gràfic d'aquesta; colors, fonts, formes ... etc. Tenim dues opcions de canvi: una és el *Look&Feel* anomenat *NimRod* (que podem veure en la figura 4.9) i l'altre és l'estil metàlic que anomenarem *DefaultLookandFeel* (que podem veure en la figura 4.10). Per variar entre els dos dissenys caldrà que seleccionem la casella d'un o de l'altre disseny tal i com es veu en la figures anteriors 4.9 i 4.10.

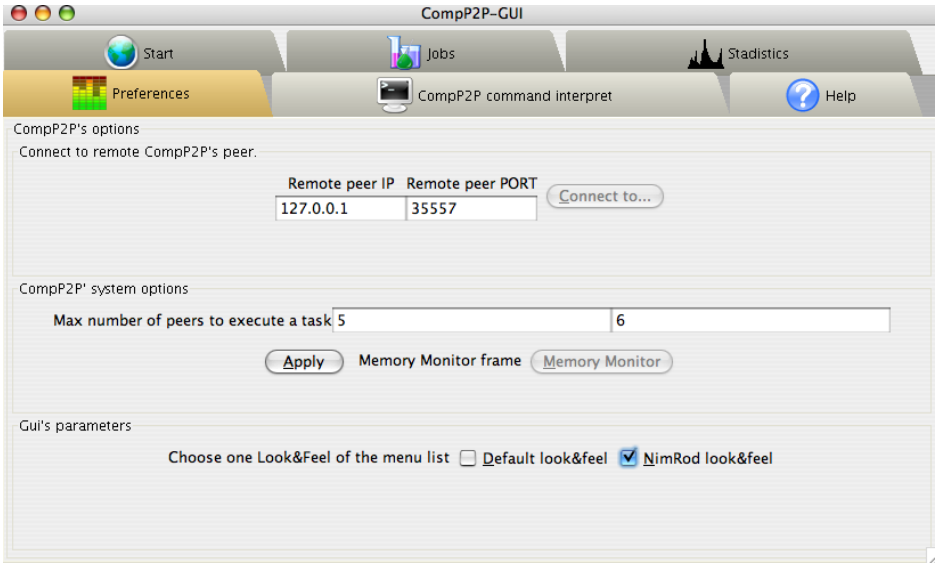

Figura 4.9: CompP2P-GUI pestanya *Preferences*; *Look&Feel NimRod*.

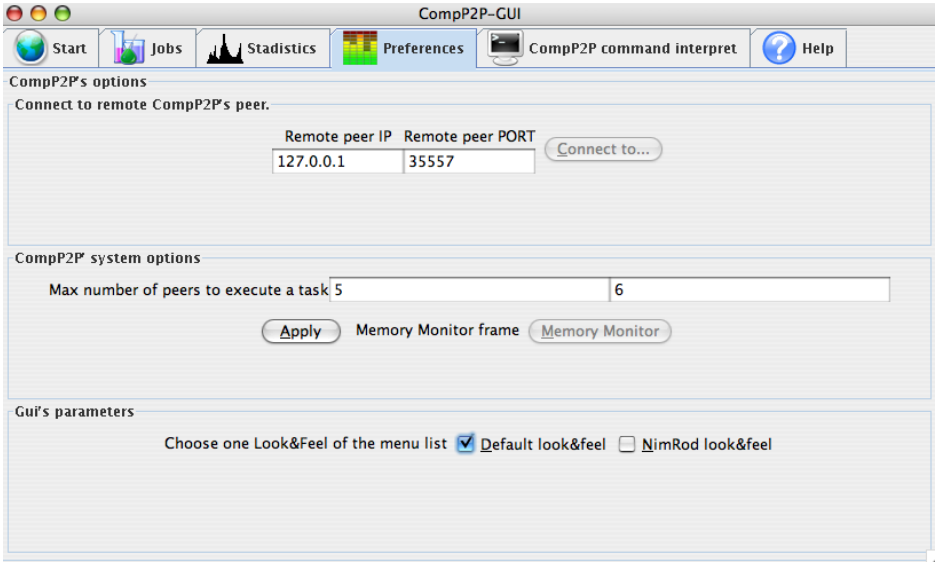

Figura 4.10: CompP2P-GUI pestanya *Preferences*; *Look&Feel Metal*.

4.7 Tab Terminal

Per donar més possibilitats al programador i més interacció amb el sistema s'utilitzen unes llibreries d'utilitats que faciliten la comunicació i l'obtenció d'informació de *CompP2P*. Aquest mòdul està format per una sola classe: **CompP2PUtils** que va integrat en el projecte inicial.

CompP2PUtils és l'única classe del mòdul i té com a objectiu implementar un joc d'utilitats per gestionar el sistema *CompP2P*. En la interfície gràfica s'ha volgut integrar aquesta eina integrant-la en el disseny gràfic en la pestanya que anomenarem **Terminal**. El seu funcionament és senzill; es tracta d'una

àrea de text de tamany gran fons negre on es mostra la informació de les peticions que l'usuari introdueix en la casella inferior de fons blanc.

L'usuari escriu una ordre que sigui acceptada per **CompP2P**[2] i es pitja la tecla *Enter*, a continuació el sistema processa l'ordre i mostra la informació per l'àrea de text gran que es troba en el centre, tal i com es mostra en la figura 4.11.

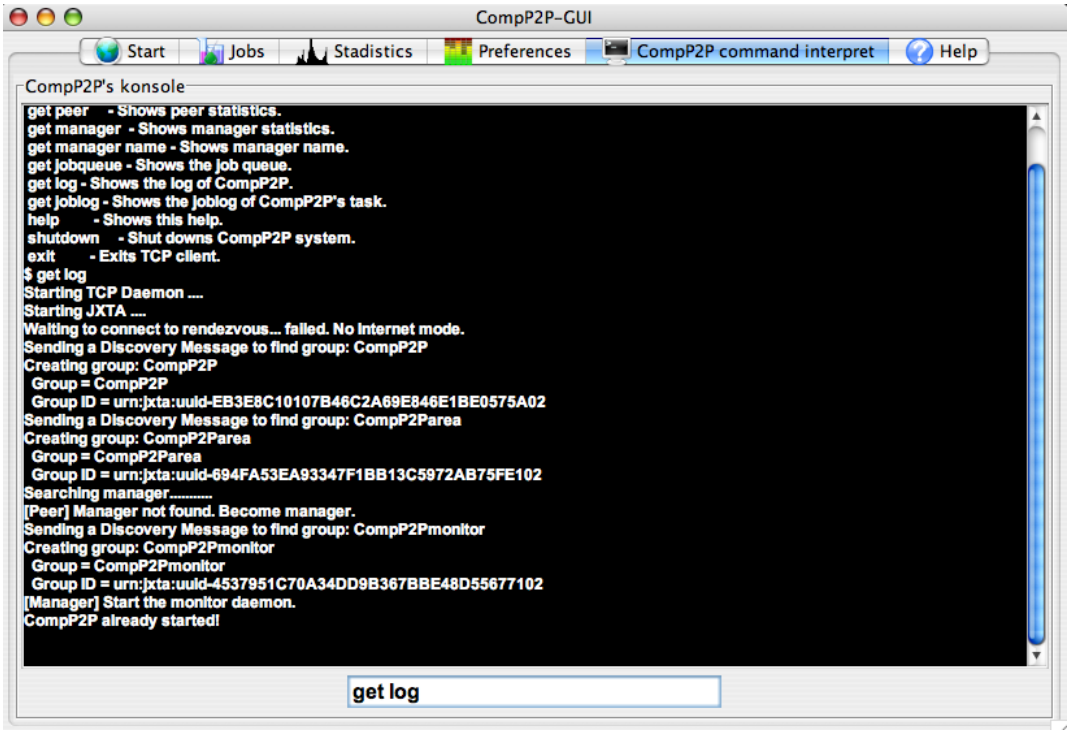

Figura 4.11: CompP2P-GUI pestanya *Terminal*

4.8 Tab Help

És la pestanya que conté informació referent a les comandes disponibles per la **Terminal**, informació dels autors del projecte *CompP2P* i *CompP2P-GUI*, així com la documentació del projecte que es mostra fent ús d'un comprimit *.jar* anomenat *jpedalSTD.jar*[11]. Una captura de pantalla d'aquest panell on es mostra informació dels autors la podem veure en la figura 4.12, mentre que en la figura **??** i 4.13 es pot veure com mostra la documentació del projecte.

[2]Les comandes que mostra el menú d'ajut de l'aplicació.

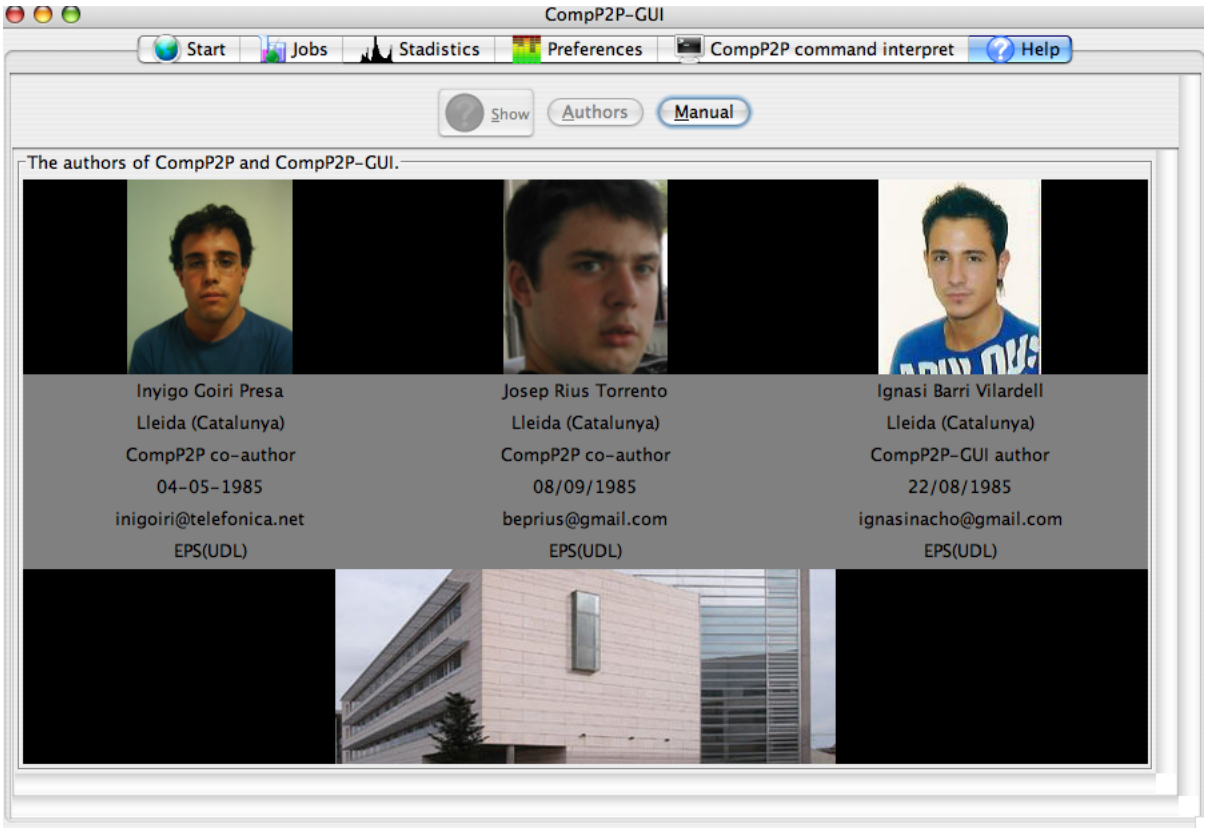

Figura 4.12: CompP2P-GUI pestanya *Help*; informació dels autors.

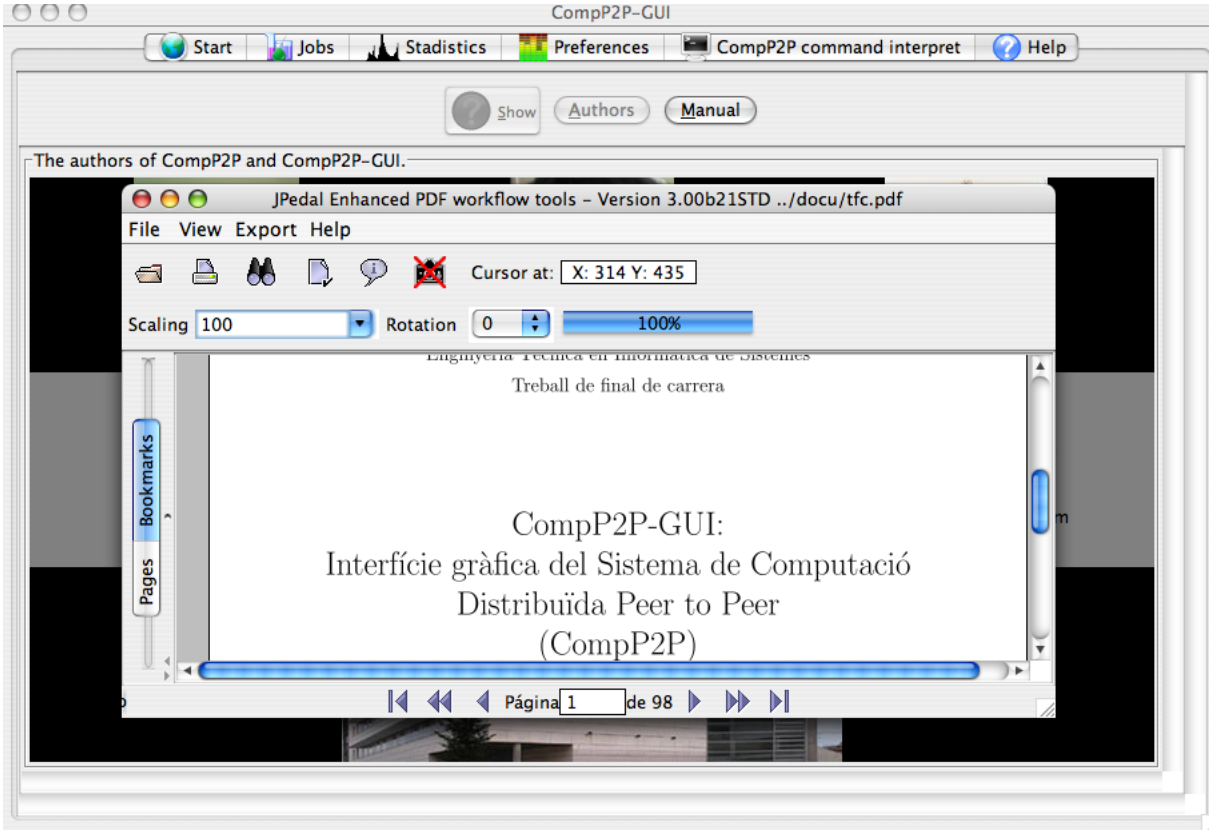

Figura 4.13: CompP2P-GUI pestanya *Help*; documentació del projecte visualitzada amb *jpedalSTD.jar*.

Capítol 5

Adaptar aplicacions al model *CompP2P*

En aquest capítol veurem com adaptar una aplicació qualsevol a un model de *CompP2P*. En dues seccions explicarem els passos a seguir per portar a terme aquesta adaptació, que consistirà en:

1. Com adaptar l'objecte d'una aplicació distribuida per *CompP2P*.
2. Com adaptar una aplicació distribuida per *CompP2P*.

5.1 Com adaptar l'objecte d'una aplicació distribuida per CompP2P

En aquesta secció s'explicarà pas a pas, i mitjançant un exemple pràctic, la manera d'adaptar el disseny d'un objecte distribuït pel sistema *CompP2P*. Podríem partir de la base que es té un objecte ja implementat, però s'ha preferit fer també l'objecte.

5.1.1 Consideracions

1. En l'objecte base:
   - Cal tenir en compte que l'objecte s'ha d'executar remotament, per tant, tota la informació necessària s'ha de passar com argument al crear-lo ja que no podem interaccionar amb ell per línia de comandes (o usant la interfície gràfica) un cop estigui en execució.
   - La classe haurà de tenir un mètode *calculate()* i un *result()*
   - Cal implementar el mètode *copia*.
2. En el proxy de l'objecte:
   - Igual que en l'objecte base cal implementar el mètode *copy*, però a més, s'ha d'implementar l'*exec()* i el *result()*, ja que están declarades com a abstractes en la superclasse Parallel.
   - D'altra banda també caldrà implementar el mètode *calculate()* ja que aquesta classe implementarà (igual que en la base) l'interfície de l'aplicació.
3. En la interfície de l'objecte:
   - Simplement declarar el mètodes *result()* i *calculate()*.

5.1.2 Exemple pràctic: Objecte de suma distribuïda *SumP2P*

Es vol dissenyar una aplicació en la que introduint dos nombres enters (x,y), aquesta faci la suma de tots els nombres que hi ha entre ells dos:

$$x + (x + 1) + (x + 2) + ... + (y - 2) + (y - 1) + y$$

L'objectiu d'això és poder dividir la suma d'una sèrie de nombres en sèries més petites i així poder distribuir la feina entre diferents *peers*. Un cop obtinguts els resultats parcials, serà l'aplicació qui s'encarregarà de juntar-los.

5.1.2.1 Especificacions de les classes

SumP2P

- **Classe:** SumP2P
- **Objectiu:** Donats dos nombres (x, y) calcular la suma dels nombres que acoten.
- **Operacions:**
  - public SumP2P(int f, int l)
    - **Crida:** SumP2P example = new SumP2P(x,y);
    - **Pre:** El primer nombre (x) ha de ser més petit que el segon (y).
    - **Post:** *example* és un objecte de la classe SumP2P buit (això és, el resultat té valor 0).
    - **Observació:** Operació constructora.
  - public void calculate() throws RemoteException
    - **Crida:** try{example.calculate();}
    - **Pre:** -
    - **Post:** Calcula la suma dels números entre x i y i la guarda en una variable.
    - **Observació:** -
  - public Object result() throws RemoteException
    - **Crida:** try{result = example.result();}
    - **Pre:** *result* és una variable de la classe Object.
    - **Post:** Mostra el resultat per pantalla i el guarda a la variable *result*.
    - **Observació:** Operació consultora.
  - public void copia(SumP2P nou)
    - **Crida:** example.copia(example2);
    - **Pre:** *example2* és una variable de la classe SumP2P.
    - **Post:** *example* és una copia de *example2*.
    - **Observació:** Operació copiadora.
- **Observacions:** Implementarà les classes *InterfaceP2P* i *Serializable*.

ProxySumP2P

- **Classe:** ProxyP2P
- **Objectiu:** Fer de classe "pont" entre les classes *Parallel* i *SumP2P*. Dit d'una altra manera, encapsular els métodes de *SumP2P* d'una manera entenedora des del punt de vista de la *Parallel*
- **Operacions:**
  - public ProxySumP2P(int f, int l)
    - **Crida:** ProxySumP2P example = new ProxySumP2P(x,y);
    - **Pre:** El primer nombre (x) ha de ser més petit que el segon (y).
    - **Post:** *example* és un objecte de la classe ProxySumP2P buit (això és, el resultat té valor 0).
    - **Observació:** Operació constructora.
  - public void calculate() throws RemoteException
    - **Crida:** try{example.calculate();}
    - **Pre:** -
    - **Post:** Calcula la suma dels nùmeros entre x i y i la guarda en una variable.
    - **Observació:** -
  - public Object result() throws RemoteException
    - **Crida:** try{result = example.result();}
    - **Pre:** *result* és una variable de la classe Object.
    - **Post:** Mostra el resultat per pantalla i el guarda a la variable *result*.
    - **Observació:** Operació consultora.
  - public void exec()
    - **Crida:** try{example.exec();}
    - **Pre:** -
    - **Post:** Calcula la suma dels nùmeros entre x i y i la guarda en una variable.
    - **Observació:** Fa el mateix que el mètode *result()*
  - protected void copy(Parallel nou)
    - **Crida:** example.copia(example2);
    - **Pre:** *example2* és una variable de la classe ProxySumP2P.
    - **Post:** *example* és una copia de *example2*.
    - **Observació:** Operació copiadora.
- **Observacions:** a més d'implementar les classes *InterfaceP2P* i *Serializable*, serà derivada de *Parallel*.

InterfaceSumP2P

- **Classe:** InterfaceSumP2P
- **Objectiu:** Suplir la carència de l'herència múltiple que té java.
- **Operacions:**
  - Object result() throws RemoteException
    - **Observació:** Abstracta
  - void calculate() throws RemoteException
    - **Observació:** Abstracta
- **Observacions:** No es necessari declarar explícitament cada funció com abstracta ja que el sol fet de ser una *Interface* en comptes d'una *class* implica que tots els mètodes són abstractes.

5.1.2.2 Implementació de les classes

Tenint en compte l'especificació i totes les consideracions de l'apartat anterior, només queda implementar les tres classes formant així el paquet sump2p.

SumP2P Com a classe principal del "paquet", *SumP2P* és la que implementa realment els diferents mètodes:

1. El mètode constructor simplement inicialitza les variables locals amb els atributs pasat com arguments en el moment de la creació del objecte.
2. El mètode *calculate* fa un bucle de $last - first$ voltes sumant a cadascuna el nou valor al resultat parcial.
3. El mètode *result* simplement retorna la variable resultat com un *Integer*
4. El mètode *copia*, imposat per la superclasse *Parallel* fa la copia argument a argument de totes les variables del objecte.

Així doncs, *sumP2P* queda de la següent manera:

```
package sump2p;

import parallel.*;
import java.io.*;

public class SumP2P implements InterfaceSumP2P, Serializable{
  private int first;
  private int last;
  private int result;

  public SumP2P(int f, int l){
    first = f;
```

```
    last = l;
    result = 0;
  }

  public void calculate() throws RemoteException{
    int i;
    for(i=first; i<=last; i++){
      result=result+i;
    }
  }

  public Object result() throws RemoteException{
    return new Integer(result);
  }

  public void copia(SumP2P nou){
    first = nou.first;
    last = nou.last;
    result = nou.result;
  }
}
```

ProxySumP2P La classe *ProxySumP2P* encapsula els mètodes de la clase *sumP2P*, afegint un control sobre l'objecte per assegurar-se de que no estigui sent cridat des d'algun altra aplicació. A més, també implementa el mètode *copy*, que a part de copiar l'únic atribut al nou objecte pasat per parametres, també fa una crida al mètode *copia* de la classe *sumP2P*:

```
package sump2p;

import parallel.*;
import java.io.*;

public class ProxySumP2P extends Parallel implements InterfaceSumP2P, Serializable{
  SumP2P obj;

  public ProxySumP2P(int f, int l){
    obj = new SumP2P(f,l);
  }

  public void calculate() throws RemoteException{
    if(!isHere){throw new RemoteException();}
      else{obj.calculate();}
  }

  public Object result() throws RemoteException{
    if(!isHere){throw new RemoteException();}
```

```
      else{return obj.result();}
  }

  public void exec(){
    try{this.calculate();}
    catch(Exception e) {e.printStackTrace();}
  }

  protected void copy(Parallel nou){
    SumP2P c = ((ProxySumP2P)nou).obj;
    obj.copia(c);
  }
}
```

InterfaceSumP2P Per últim, en la interfície es declaren els dos mèteodes que forçadament ha d'implementar tant la classe *sumP2P* com *ProxySumP2P*:

```
package sump2p;
import parallel.*;

public interface InterfaceSumP2P{
  Object result() throws RemoteException;
  void calculate() throws RemoteException;
}
```

5.2 Com adaptar una aplicació distribuida per CompP2P

Seguint amb l'exemple de l'apartat anterior 5.1, el següent pas és implementar l'aplicació que calculi la suma dels primers "x" números. Més concretament, l'aplicació calcularà la suma dels 100.000 primers nombres.

Un cop explicats els objectius d'aquesta aplicació, es detallen els pasos de la implementació:

1. Primer de tot, cal importar les llibreries que utilitzarem:

   ```
   import cat.udl.eps.compp2p.sump2p.ProxySumP2P;
   import cat.udl.eps.compp2p.utils.CompP2PUtils;
   ```

2. En quant als atributs:

   (a) És crea un objecte CompP2PUtils per obtenir el número de peers que formen part del sistema (en el moment de l'execució d'aquesta aplicació) mitjançant el mètode *getPeers()*.

   ```
   CompP2PUtils stats = new CompP2PUtils();
   int peers=stats.getPeers();
   ```

   (b) Es declara els números que volem sumar:

   ```
   int total=100000;
   ```

(c) Es crea un vector de ProxySumP2P de "peers" posicions, per guardar els resultats parcials i s'inicialitza assignant 100.000/peers a cada objecte del array.

```
ProxySumP2P example[]=new ProxySumP2P[num];
```

3. Finalment, amb l'ajut de dos bucles, es recorren totes les possicions d'*example[]* cridant amb el primer al mètode *launch()* i amb el segon a *stop()*. Com ja s'ha explicat en les anteriors especificacions, és necessari cridar al mètode *stop()* si s'ha cridat prèviament al *launch()*. Això és degut que durant l'execucio d'aquest útlim mètode, l'objecte esdevé bloquejat per evitar problemes de sincronització, i per tant és imprescindible desbloquejar-lo amb *stop()* abans de poder interaccionar amb ell.

Per tant, l'aplicació queda de la següent manera:

```
import cat.udl.eps.compp2p.sump2p.ProxySumP2P;
import cat.udl.eps.compp2p.utils.CompP2PUtils;

class Aplicacio{
  public static void main(String args[]){
    CompP2PUtils stats=new CompP2PUtils();
    int peers=stats.getPeers();
    int total=100000;

    System.out.println("We have " + peers + " peers, we will do " + peers + " jobs.");

    ProxySumP2P example[]=new ProxySumP2P[peers];

    try{
      for(int i=0; i<peers; i++){
        example[i]=new ProxySumP2P(i*(total/peers)+1, ((i+1)*(total/peers)) );
        example[i].launch("sump2p.jar");
      }

      for(int i=0; i<peers; i++){
        example[i].stop();
      System.out.println("Result number " + i + ": " + ((Integer)example[i].result()).intValue())
      }

    }
    catch(Exception e){System.out.println("ERROR!");}
  }
}
```

5.3 Compilació de l'aplicació adaptada

En aquesta secció s'indicarà com compilar aquesta nova aplicació adaptada, ja que requereix d'unes llibreries o arxius *.jar*[1] que es troben en una ruta de l'aplicació específica. Així doncs, anem a veure quina

[1]Els fitxers *.jar* són del tipus *Java ARchive*, és un fitxer *.zip* que s'usa per distribuïr un conjunt de classes *Java* que es troben emmagatzemades i associades a una metadada que pot constituir un programa.

comanda hem d'executar per compilar una classe adaptada.

Seguint amb l'exemple, caldrà que ens situem dins de la carpeta de la nova classe creada, i teclegem en la consola el següent:

javac -classpath ../lib/parallel.jar:../lib/utils.jar:NOM_CLASSE_ADAPTADA.jar:. Aplicacio.java

On el *parallel.jar* i *utils.jar* són necessaris per totes les aplicacions i el *NOM_CLASSE_ADAPTADA.jar* és un fitxer amb una estructura semblant a la que es mostra en la figura 5.1 on es veu l'estructura del fitxer *Fibonacci.jar*.

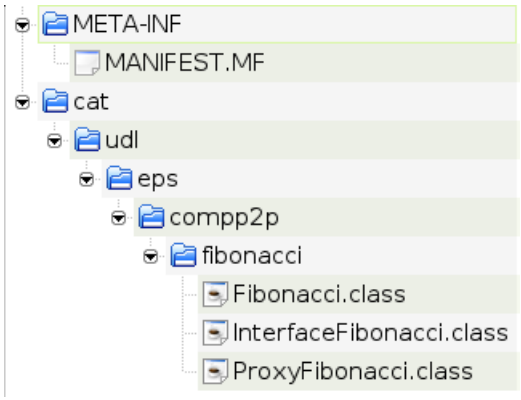

Figura 5.1: Estructura del fitxer .jar per l'aplicació adaptada *Fibonacci*

I *Aplicacio.java* és l'aplicació adaptada que es mostra en la secció 5.2.

Capítol 6

Conclusions

Dels objectius plantejats en aquest treball en un principi és pot dir que tots ells s'han assolit amb molt bons resultats. Seguidament es fa una repassada a tots ells.

El disseny de l'aplicació ha estat desenvolupat modularment; separa les diferents tasques i afavoreix la modificació dels diferents components de *CompP2P-GUI*. Juntament amb la documentació, ofereix a futurs programadors en aquest entorn unes facilitats afegides.

La "multiplataformitat" ha estat aconseguida amb escreix, *CompP2P-GUI* és totalment independent del sistema operatiu. En gran mesura aquest fet ve donat per l'entorn de programació, cal destacar, també, que Java 1.5 (*Swing*) ha aconseguit satisfer les necessitats del projecte.

La instal·lació del programa és molt senzilla i no necessita cap modificació en el nostre sistema, ja siguin arxius o paràmetres del mateix. Només es precisa la instal·lació de la màquina virtual de Java Sun en el nostre sistema. Aquestes apreciacions són d'important rellevància per l'usuari.

Pel que fa a la documentació del projecte ha estat més que satisfactòria, fa un seguiment de tot el procés de disseny, les característiques del sistema i la motivació de cadascuna d'elles. D'altra banda, molt lluny de ser totalment teòrica, s'ha acompanyat aquesta documentació amb alguns exemples pràctics paral·lels a l'explicació d'alguns apartats. Un altre dels seus punts forts, ha estat la documentació del codi, on s'ha fet una delcaració de la seva API fent ús de l'eina *javadoc*, la qual aconsegueix uns resultats de caire professional.

Un cop ja s'ha parlat de l'entorn de *CompP2P-GUI* cal analitzar l'aplicació pròpiament dita.

Comentar que les pretencions inicials de senzillesa i eficàcia que s'havien marcat, s'han aconseguit portant a terme un disseny gràfic molt amigable a la vegada que funcional, la qual cosa és un punt fonamental d'aquest projecte. També, comentar que és tracta d'una primera presa de contacte amb *CompP2P* (ja que s'han modificat parts d'aquests), a la vegada que l'inici d'un camí, en el qual s'ha demostrat que és necessàri dotar d'un entorn gràfic a una aplicació com és *CompP2P*.

Comentar que, evidentment, hi ha moltes coses que podríen estar incloses en el sistema, i d'altres que potser algun altre desenvolupador hauria trobat no tan imprescindibles; l'objectiu, però, ha estat balancejar la càrrega d'aquestes "coses" prescindibles, enfocant tot el ventall de possibilitats a aquelles que ens permeten dotar d'informació addicional a la que ja ens mostra *CompP2P* (monitor de memòria, variables del sistema... etc). D'aquesta manera, s'ha obtingut un sistema funcional i força complet.

Per concloure podríem dir que els resultats obtinguts han estat satisfactoris, ja que uns dels principals objectius era fer una base sòlida facilitar-ne les futures modificacions; projecte en què no dubtaria embarcar-m'hi nòvament.

Capítol 7

Treball futur

Aquest projecte, és el que es coneix en termes econòmics "*ampliar la quota de mercat*", és a dir, aquest projecte ha fusionat una aplicació molt usable i útil amb una eina per tal que la utilitat d'aquesta creixi de forma considerable, de tal manera que aquesta interfície gràfica obre les portes a usuaris menys expert en el camp de la computació distribuïda, o facilita les coses a les persones que desconeixen el funcionament de *CompP2P*.

Aquest treball, és una simple presa de contacte amb el món de les interfícies gràfiques i els programes *p2p*. També ha servit per denotar que la creació d'una interfície gràfica és molt interessant, o determinant (per que no), alhora de "vendre" un producte.

Així doncs, tenim una interfície gràfica que bona part de la seva evolució va lligada al fet que *CompP2P* també es vagi millorant, introduint noves funcionalitats que, més tard, seran instanciades amb una àrea de text, un botó, un menú ... de l'interfície gràfica d'usuari.

D'altra banda, hem de ésser crítics amb el treball realitzat, i tot i que l'aplicació cobreix amb escreix les funcionalitats de l'aplicació nativa (*CompP2P*) i se n'han introduït d'altres que tracten conceptes interessants(el menú de conversió d'una aplicació al model *CompP2P*, la informació de variables locals del sistema, modificació d'alguns aspectes de *CompP2P*, la connexió remota ... etc), també és evident, que l'àmbit al qual pertany *CompP2P*, dóna un ventall de possibilitats força ampli per treballar la interfície gràfica indepentment del que la capa base del programa evolucioni; estem parlant de la manera de realitzar estadístiques, és a dir, la inclusió de gràfiques segons els resultats, realitzar mitges amb els temps d'execució obtinguts, càlcul de la potència de còmput ... etc, que creiem que serà força interessant tractar-ho en un treball futur, i que estem convençuts que es portarà a terme ben aviat.

Finalment, també es podria tractar en una versió futura de l'aplicatiu, aspectes com:

- Implementació interfase RMI entre monitor i sistema P^3CoDi.
- Migrar el monitor a un entorn Web (pot ser com un applet Java).
- Creació d'una *BD* on es desarien els resultats de les execucions que els diferents usuaris porten a terme, aplicacions que s'executen ... etc.
- Inclusió d'un monitor de rendiment del microprocessador (tal i com hem fet amb el monitor de memòria).

Bibliografia

[1] Java 1.4.2 API `http://java.sun.com/j2se/1.4.2/docs/api/`

[2] Instalació de Java 1.5 en Linux `http://www.java.com/en/download/help/5000010500.xml`

[3] Instalació de Java 1.5 en Linux `http://www.java.com/en/download/linux_manual.jsp`

[4] Instalació de Java 1.5 en Windows `http://www.java.com/en/download/help/5000010300.xml`

[5] JXTA 2.3.7 API `http://platform.jxta.org/nonav/java/api/index.html`

[6] JXTA Programmer's Guide `http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf`

[7] JXTA Book `http://www.brendonwilson.com/projects/jxta-book/`

[8] JNGI `http://jngi.jxta.org/`

[9] Wikipedia `http://es.wikipedia.org/`

[10] Memòria del *TFC CompP2P*, Iñigo Goiri Presa i Josep Rius Torrentó, 2006

[11] JPEDAL Project `http://www.jpedal.org/`

[12] The Swing tutorials `http://java.sun.com/docs/books/tutorial/uiswing/`

[13] N. Drost, R.V. van Nieuwpoort, H. Bal, "Simple Locality-Aware Co-allocation in Peer-to-Peer Supercomputing", Proc. of the 6th IEEE Int. Symposium on Cluster Computing and Grid Workshops (CCGRIDW?06), 2006.

[14] A. Auvinen, M. Vapa, M. Weber, N. Kotilainen and J. Vuori, "Chedar: Peer-to-Peer Middelware", Proc. of the 20th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS'06), 2006.

[15] I. Foster and A. Iamnitchi, "On death, taxes, and the convergence of peer-to-peer and grid computing", Proc of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), 2003.

[16] M. Venkateswara Reddy, A. Vijay Srinivas, Tarun Gopinath, D. Janakiram, "Vishwa: A reconfigurable P2P middleware for Grid Computations", Proc. of the International Conference on Parallel Processing (ICPP'06) pp. 381-390, 2006.

[17] D. Talia, P. Trunfio, "Towards a synergy between P2P and Grids", IEEE Internet Comput. 7 (4), pp. 94-96, 2003.

[18] Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proceedings of ACM SIGCOMM, San Diego, CA, USA, 2001.

[19] A. Gupta, D. Agrawal, A. E. Abbadi, "Distributed Resource Discovery in Large Scale Computing," SAINTW 2005.

[20] F. Mao, H. Jin, D. Zou, B. Chen, L. Qi, "QoS Oriented Dynamic Replica Cost Model for P2P Computing", Proc. of the 25th IEEE Int. Conference on Distributed Computing Systems Workshops (ICDCSW?05), 2005.

[21] Zhiyong Xu Bhuyan, "Effective Load Balancing in P2P Systems", Sixth IEEE International Symposium on Cluster Computing and the Grid, pp. 81- 88, 2006.

[22] Y. Murata, T. Inaba, H. Takizawa, H. Kobayashi, "A distributed and cooperative load balancing mechanism for large-scale P2P systems", Proc. of the Int. Symposium on Applications and the Internet Workshops (SAINTW?06), 2006.

[23] J. Hu and R. Klesftad, "Decentralized Load Balancing on Unstructured Peer-2-Peer Computing Grids", Int. Symposium on Applications and the Internet Workshops (SAINTW?06), 2006.

[24] C. Anglano, "Interceptor: Middleware-level Application Segregation and Scheduling for P2P Systems", 20th International Parallel and Distributed Processing Symposium, 2006.

[25] N. Griffiths, K.-M. Chao, and M. Younas, "Fuzzy Trust for Peer-to-Peer systems", 26th International Conference on Distributed Computing Systems (ICDCS 2006), 2006.

[26] Wei Wu, Phu Dung Le: "An Efficient and Secure Code Sharing for Peer-to-Peer Communications", International Conference on Information Technology: New Generations, pp. 476-481, 2006.

[27] A. Mondal, M. Kitsuregawa, "Privacy, Security and Trust in P2P environments: A Perspective", 3rd Int. Workshop on P2P Data Management, Security and Trust, pp. 682-686, 2006.

[28] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-peer computing", Technical Report HPL-2002-57, HP Lab, 2002.

[29] Article sobre *P2P* `http://es.wikipedia.org/wiki/P2P`

[30] NimRod *Look & Feel* `http://personales.ya.com/nimrod/index.html`

[31] MemoryMonitor *Sun class* `http://www.informit.com/guides/content.asp?g=java&seqNum=249&rl=1`

Apèndix A

Codi CompP2PGUI

A continuació es mostrarà el codi de la classe principal del programa *CompP2P-GUI*: *CompP2PGUI*.

CompP2PGUI

```
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import javax.swing.JTextArea;
import java.lang.String;
import cat.udl.eps.compp2p.utils.CompP2PUtils;
import com.nilo.plaf.nimrod.NimRODLookAndFeel;
import java.io.LineNumberReader;
import java.net.Socket;
import java.lang.System;
import javax.swing.SwingUtilities;


/**
 * Is the principal Class, match the rest of the class to the principal Frame.
 * @author Ignasi Barri Vilardell
 * @version 1.0
 */
public class CompP2PGUI extends JFrame {
     JPanel connect, term, state, prefi;
     JScrollPane jelp;
     JTabbedPane tabbedPane, work;
```

```
boolean switchs, remote=false, contador=false;
Graphics g;

private Inici inici;
private Job job;
private Statistics statistics;
private Preferenses preferenses;
private Terminal terminal;
private Help help;

private CompP2PUtils utils;


public CompP2PGUI() {

     super("CompP2P-GUI EPS Project");

     tabbedPane = new JTabbedPane();

     //Servidor i Connectar
     ImageIcon serversicon = new ImageIcon("gui/img/32x32/servers.png");
     inici=new Inici(this);
     connect=inici.create();
     tabbedPane.addTab("Start", serversicon, connect);

     //Treballs, recents i altres
     ImageIcon jobicon = new ImageIcon("gui/img/32x32/job.png");
     job = new Job(this);
     work=job.create();
     tabbedPane.addTab("Jobs", jobicon, work);

     //Estadistiques (Peer, Manager, General,....)
     ImageIcon stadicon= new ImageIcon("gui/img/32x32/stads.png");
     statistics = new Statistics(this);
     state=statistics.create();
     tabbedPane.addTab("Stadistics", stadicon, state);

     ImageIcon preferencesicon = new ImageIcon("gui/img/32x32/conf.png");
     preferenses = new Preferenses(this);
     prefi = preferenses.create();
     tabbedPane.addTab("Preferences", preferencesicon, prefi);

     ImageIcon terminalicon = new ImageIcon("gui/img/32x32/client.png");
     terminal = new Terminal(this);
     term = terminal.create();
     tabbedPane.addTab("CompP2P command interpret", terminalicon, term);
```

```
          ImageIcon helpicon = new ImageIcon("gui/img/32x32/help.png");
          help = new Help(this);
          jelp = help.create();
          tabbedPane.addTab("Help", helpicon, jelp);

          getContentPane().add(tabbedPane);

          utils=new CompP2PUtils();
          }

/**
 * Get CompP2P utils.
 * @see getInici()
 * @see getJob()
 * @see getStatistics()
 * @see getPreferences()
 * @see getTerminal()
 * @see getHelp()
 * @version 1.0
 */
public CompP2PUtils getUtils(){
          return utils;
}

/**
 * Get "Inici" tab.
 * @see getUtils()
 * @see getJob()
 * @see getStatistics()
 * @see getPreferences()
 * @see getTerminal()
 * @see getHelp()
 * @version 1.0
 */
public Inici getInici(){
          return inici;
}

/**
 * Get "Job" tab.
 * @see getUtils()
 * @see getInici()
 * @see getStatistics()
 * @see getPreferences()
 * @see getTerminal()
```

```
 * @see getHelp()
 * @version 1.0
 */
public Job getJob(){
          return job;
}

/**
 * Get "Statistics" tab.
 * @see getUtils()
 * @see getInici()
 * @see getJob()
 * @see getPreferences()
 * @see getTerminal()
 * @see getHelp()
 * @version 1.0
 */
public Statistics getStatistics(){
          return statistics;
}

/**
 * Get "Preferences" tab.
 * @see getUtils()
 * @see getInici()
 * @see getJob()
 * @see getStatistics()
 * @see getTerminal()
 * @see getHelp()
 * @version 1.0
 */
public Preferenses getPreferenses(){
          return preferenses;
}

/**
 * Get "Terminal" tab.
 * @see getUtils()
 * @see getInici()
 * @see getJob()
 * @see getStatistics()
 * @see getPreferences()
 * @see getHelp()
 * @version 1.0
 */
public Terminal getTerminal(){
```

```
          return terminal;
}

/**
 * Get "Help" tab.
 * @see getUtils()
 * @see getInici()
 * @see getJob()
 * @see getStatistics()
 * @see getPreferences()
 * @see getTerminal()
 * @version 1.0
 */
public Help getHelp(){
          return help;
}

/**
 * Repaint all components of GUI.
 * @version 1.0
 */
public void repaintAll() {
          tabbedPane.updateUI();
}

/**
 * Set the principal buttons of the "GUI" to "Enable" mode.
 * @see setComponentsDisconnect()
 * @version 1.0
 */
public void setComponentsConnect() {
          getInici().getConnectButton().setEnabled(false);
          getInici().getDisconnectButton().setEnabled(true);
          getJob().getAppButton().setEnabled(true);
          getStatistics().getPeerInfoButton().setEnabled(true);
          getStatistics().getManagerInfoButton().setEnabled(true);
          getPreferenses().getApplyButton().setEnabled(true);
          getPreferenses().getRemoteButton().setEnabled(false);
          getHelp().getHelpButton().setEnabled(true);
          switchs=false;
          remote=true;
}

/**
 * Set some components of the "GUI" to "Disconnect" mode.
 * @see setComponentsConnect()
```

```
 * @version 1.0
 */
public void setComponentsDisconnect() {
          getInici().getConnectButton().setEnabled(true);
          getInici().getDisconnectButton().setEnabled(false);
          getJob().getAppButton().setEnabled(false);
          getStatistics().getPeerInfoButton().setEnabled(false);
          getStatistics().getManagerInfoButton().setEnabled(false);
          getPreferenses().getApplyButton().setEnabled(false);
          getPreferenses().getRemoteButton().setEnabled(true);
          getHelp().getHelpButton().setEnabled(false);
          switchs=true;
          remote=false;
}

/**
 * Create and show CompP2P-GUI.
 * @version 1.0
 */
public static void main(String[] args) {
          JFrame frame = new CompP2PGUI();
          Toolkit tk = Toolkit.getDefaultToolkit();
          Dimension d = tk.getScreenSize();
          int resalt = d.height;
          int resample = d.width;
          frame.setSize(resample/2, (resalt*7)/10);
          frame.setLocation(resample/4, resalt/4);
          frame.setTitle("CompP2P-GUI");
          frame.addWindowListener(new WindowAdapter() {
               public void windowClosing(WindowEvent e) {System.exit(0);}
          });
          JFrame.setDefaultLookAndFeelDecorated(true);
          frame.setVisible(true);

        }
}
```